RELIABLE DATA DELIVERY IN WIRELESS SENSOR NETWORKS

A Thesis Submitted to the College of
Graduate Studies and Research
In Partial Fulfillment of the Requirements
For the Degree of Master of Science
In the Department of Computer Science
University of Saskatchewan
Saskatoon, Saskatchewan, Canada

By

Bofu Yang

PERMISSION TO USE

In presenting this thesis in partial fulfilment of the requirements for a Postgraduate degree from the University of Saskatchewan, I agree that the Libraries of this University may make it freely available for inspection. I further agree that permission for copying of this thesis in any manner, in whole or in part, for scholarly purposes may be granted by the professor or professors who supervised my thesis work or, in their absence, by the Head of the Department or the Dean of the College in which my thesis work was done. It is understood that any copying or publication or use of this thesis or parts thereof for financial gain shall not be allowed without my written permission. It is also understood that due recognition shall be given to me and to the University of Saskatchewan in any scholarly use which may be made of any material in my thesis.

Requests for permission to copy or to make other use of material in this thesis in whole or part should be addressed to:

Head of the Department of Computer Science
University of Saskatchewan
Saskatoon, Saskatchewan, Canada
S7N 5C9

ABSTRACT

Wireless sensor networks (WSN) have generated tremendous interest among researchers these years because of their potential usage in a wide variety of applications. Sensor nodes are inexpensive portable devices with limited processing power and energy resources. Sensor nodes can be used to collect information from the environment, locally process this data and transmit the sensed data back to the user.

This thesis proposes a new reliable data delivery protocol for general point-to-point data delivery (unicasting) in wireless sensor networks. The new protocol is designed that aims at providing 100% reliability when possible as well as minimizing overhead and network delay. The design of the new protocol includes three components. The new protocol adopts a NACK-based hop-by-hop loss detection and recovery scheme using end-to-end sequence numbers. In order to solve the single/last packet problem in the NACK-based approach, a hybrid ACK/NACK scheme is proposed where an ACK-based approach is used as a supplement to the NACK-based approach to solve the single/last packet problem. The proposed protocol also has a new queue management scheme that gives priority to new data. By introducing the idea of a Ready_Bit and newer packet first rule in the transmission queue, nodes can detect and recover lost packets in parallel with the normal data transmission process.

The performance of the new protocol is tested in a Crossbow MicaZ testbed. Experimental results show that the new protocol performs well under various system and protocol parameter settings.

ACKNOWLEDGMENTS

I would like to express my sincere appreciation to my thesis supervisor Dr. Derek Eager for all his invaluable guidance, support and encouragement. His perpetual energy and enthusiasm in research had motivated all his advisees, including me. In addition, he was always accessible and willing to help his students with their research. As a result, research life became smooth and rewarding for me. It has been my great pleasure and honor to have worked with him. I would also like to thank the members on my supervisory committee, Dr. Dwight Makaroff and Dr. Ralph Deters, as well as my external examiner, Dr. Li Chen, for their helpful comments and constructive suggestions.

I would also like to express my gratefulness to all the faculty, staff and graduate students in the Department of Computer Science, for their caring and support during my study at the University of Saskatchewan.

Finally, I would like to thank my family and friends for their encouragement and help, especially my parents who have devoted all their efforts in me, I am grateful for their continuing understanding and countless support throughout my entire life.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF ACROYNMS

ACK	Acknowledgement
AIMD	Additive Increase Multiplicative Decrease
ARQ	Automatic Repeat Request
CODA	Congestion Detection and Avoidance
DTSN	Distributed Transport for Sensor Networks
ESRT	Event-to-Sink Reliable Transport
FEC	Forward Error Correction
FIFO	First In First Out
IP	Internet Protocol
MDS	Minimum Dominating Set
NACK	Negative Acknowledgement
nesC	Network Embedded Systems C
PALER	Push Aggressively with Lazy Error Recovery
PORT	Price-Oriented Reliable Transport
PSFQ	Pump Slowly, Fetch Quickly
QoS	Quality of Service
RBC	Reliable Bursty Convergecast
RCRT	Rate-Controlled Reliable Transport
ReInForM	Reliable Information Forwarding
RMST	Reliable Multi- Segment Transport
RTS/CTS	Request to Send / Clear to Send
RTT	Round Trip Time
TCP	Transmission Control Protocol
WSN	Wireless Sensor Network

CHAPTER 1
INTRODUCTION

A wireless sensor network (WSN) consists of a group of self-organizing, lightweight sensor nodes that are used to cooperatively monitor physical or environmental conditions. Commonly monitored parameters include temperature, sound, humidity, vibration, pressure and motion [40]. Each sensor node in a WSN is equipped with a radio transmitter, several sensors, a battery unit and a microcontroller. Although WSN research was initially motivated by military applications, wireless sensor networks are now used in many industrial and public service areas including traffic monitoring, weather conditions monitoring, video surveillance, industrial automation and healthcare applications [1]. Because of the size and cost constraints on sensor nodes, they are limited by energy, bandwidth, memory and other resources. Any protocol design for WSNs needs to consider the limitations of sensor nodes carefully. This thesis proposes a new hop-by-hop NACK-based reliable data delivery protocol that aims to provide high reliability with minimal delay and overhead. The rest of this chapter is organized as follows, Section 1.1 and Section 1.2 provide overviews of WSNs and the reliable data delivery in WSNs, respectively. The motivation of this thesis is described in Section 1.3. Section 1.4 describes the contributions. Section 1.5 presents the thesis organization.

1.1 Wireless Sensor Networks Overview

WSNs have generated tremendous interest among researchers these years because of their potential usage in a wide variety of applications. Sensor nodes are inexpensive portable devices with limited processing power and energy resources. Sensor nodes can be used to collect information from the environment, locally process this data and transmit the sensed data back to the user.

Sensor nodes consist of five main components [17]: a computing unit, a communication unit, a sensing unit, a memory unit, and a power supply unit. The

computing unit consists of a microprocessor. The microprocessor is responsible for managing the communication protocols, processing collected data from the on-board sensors, and performing the power management. Each sensor node has a single communication unit that is able to transmit and receive packets. This unit combines the functionality of both transmitter and receiver. The communication frequencies of the sensor nodes are between 433 MHz (in some early generations of sensor nodes) and 2.4 GHz (the most commonly used frequency) [2]. The communication unit has four operational states: transmit, receive, idle and sleep. A sensing unit is usually a sensor board that consists of one or more sensors. Sensors must have extremely low power consumption. Some commonly used sensors are temperature sensor, humidity sensor, light sensor, barometer, 2-axis accelerometer, microphone, and GPS receiver. There are two types of memory units based on different needs for storage in a sensor node. The microprocessor itself contains some on-chip memory used to store system software. There is also typically flash memory available where users can store their own applications and data. The power unit provides power to other four units described above. In the MicaZ mote, for example, it consists of two AA batteries, either rechargeable or non-rechargeable [2]. Although all sensing, computing and communication operations consume energy, data communication requires more energy than sensing and computing. Thus, reducing data communication between sensor nodes can improve the energy efficiency and extend the lifetime of sensor networks.

As shown in Figure 1.1, typical wireless sensor networks consist of multiple sensor nodes deployed in the sensing field, and one or several sinks nodes at which data is collected and which have external network connectivity. Sensor networks in many applications are deployed without pre-defined structure and left unattended to perform multiple monitoring or tracking tasks. A WSN is able to self-configure its operation and manage its connectivity. A WSN is also able to tolerate malfunctioning nodes and integrate new nodes in the network since node failure is common in WSN applications [12]. Because of the limited power and transmission range in a large sensor network, the communication between sensor nodes must be multihop. Data from a source sensor node relayed by a number of intermediate nodes before it reaches the final destination. Collaboration between sensor nodes and in-network processing are necessary in sensor

networks since a single node may not have all the data concerning some event of interest [8] [13]. In-network processing can also reduce the number of packets transmitted in the network by aggregating similar data together and thus reducing the power consumption.

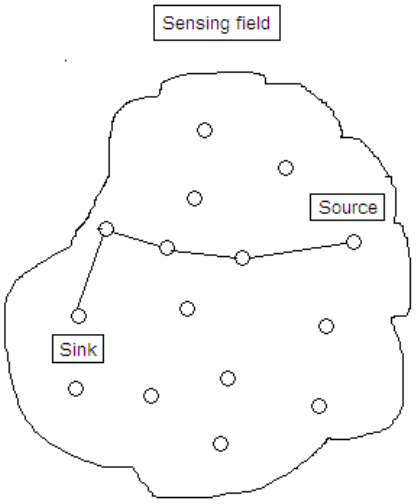


Figure 1.1 Wireless Sensor Networks

Wireless sensor networks have great potential for many industrial applications. Typical WSN applications can be classified into two categories: monitoring and tracking [17]. Monitoring applications may involve periodic data collection or may be event-driven. In an event-driven application, when a certain event occurs in the sensing field, sensor nodes collect the sensor readings of that certain event and transmit them back to the sink. Those applications usually employ a very strict power management strategy due to the limited power supply of sensor nodes and long lifetime requirement of the application [9] [23][25]. For example, sensor nodes may operate most of the time in sleep mode and are only woken up by a nearby sentry node (a node that is awake all the time and monitors the sensing field) when a certain event is detected [39]. Some common WSN monitoring applications include environmental monitoring, battlefield monitoring, health monitoring, water monitoring, and greenhouse monitoring [1]. Tracking applications have different requirements than monitoring applications in that the source of an event can be mobile. Of interest is the current location of the target. Real-time communication is usually desired in tracking applications [11]. Some common tracking

applications include traffic control [33] and surveillance [10].

1.2 Reliable Data Delivery Overview

Reliable data transport is an important topic in wireless sensor networks. A reliable protocol in wireless sensor networks is a protocol that can reliably deliver packets from their sources to their destinations without packet loss. Many WSN applications require reliable data transport. For example, consider a sensor network deployed in a chemical plant to detect harmful gas. It is crucial for sensor nodes to reliably transport every sensor reading back to the sink. Other critical WSN applications such as biological monitoring, health care monitoring, and battlefield surveillance also require high end-to-end reliability. On the other hand, some applications may not require simple 100% guaranteed transmission of data packets [7]. The reason is that this guaranteed delivery is challenging and costly in terms of energy and bandwidth usage. In some circumstances, applications may only require data packets to be reliably delivered to or from a sub-region of the network or to or from a minimal number of sensor nodes that can cover the sensing area.

Due to many unique characteristics and constraints of sensor nodes, providing reliability in wireless sensor networks can be challenging. As a microelectronic device, sensor node has very limited power resources. Sensor nodes can be deployed in many non-easily accessible areas or inhospitable conditions, which make replenishment of power resources impossible. Thus, energy consumption must be considered when designing a reliable data transport protocol in wireless sensor networks. A number of strategies can be implemented in the communication protocols to reduce energy consumption in sensor networks including: reduce the data transmitting frequency, reduce the protocol and system overhead, implement data compression and aggregation schemes, implement power management mechanisms, and eliminate the transmission of redundant data [1].

Another challenge in designing a reliable protocol in wireless sensor networks is frequent node failure. Node failure in the sensor network can be the result of harsh environment, energy depletion, or system crashes. As a multihop self-organized network, malfunction of several sensor nodes can cause significant topology changes and disrupt the normal functioning of the reliable protocol in a wireless sensor network.

In order to detect and recover lost packets, a receiver feedback and sender retransmission mechanism is usually used in wireless sensor networks. There are two commonly used receiver feedback mechanisms: the ACK-based approach and the NACK-based approach. In an ACK-based approach, the receiver positively acknowledges receipt of data packets, while in a NACK-based approach, the receiver only returns feedback to the sender if it detects a packet loss. The NACK-based approach incurs less overhead than the ACK-based approach, but a common problem for the NACK-based approach is that it can not detect single/last packet loss, since packet loss detection is based on observing gaps in the packet flow. A detailed discussion of an ACK-based approach and NACK-based approach are presented in Section 2.2.2.2.

The solution of reliability can be provided in different communication layers in wireless sensor networks. Error detection in the physical layer can be helpful to achieve reliability. However, for MicaZ, a commonly-used sensor node testbed, it is hard, if not impossible to rewrite its physical layer since it has been hardcoded. The MAC layer can provide reliability mechanisms such as RTS/CTS handshake, MAC layer acknowledgement and randomized slot selection [27] [38]. RMST [29] is a good example of using a MAC layer protocol to achieve reliability. Reliability issues can also be addressed in the routing layer. One example is ReInForM [3], a reliable routing protocol that takes advantage of multipath routing to transmit redundant data packets to the receiver and thus provide reliability. Finally, reliability can be provided in the transport layer. Transport layer can implement a similar hop-by-hop error recovery scheme as in the data link layer. However, different from the data link layer, the quality and type of service provided in transport layer is negotiable. The design of a transport layer reliable protocol can be very flexible according to the specific reliability requirement of the application. Some existing transport layer reliable protocols include PSFQ [32], RBC [42], and DTSN [26].

1.3 Motivations

General speaking, the design of a data transport protocol in wireless sensor networks is focused on providing end-to-end reliability, mitigating congestion, and achieving fairness in bandwidth allocation [17]. The reliability issue in the data transport protocol

usually involves loss recovery, congestion control, or both. Most of the reliable data transport protocols either use a retransmission-based loss recovery approach or a redundant data transmission method (sending multiple copies of a data packet into the network). A detailed discussion of reliable data transport is presented in Section 2.2. As in many other types of networks, congestion in wireless sensor networks can have a significant impact on quality of service. Large numbers of lost packets, increased network latency and poor energy efficiency can be direct consequences of congestion. Congestion control protocols in WSNs concern how to detect congestion and how to mitigate congestion. Several existing congestion control protocols are discussed in Section 2.4.2.

After studying the design challenges of data transport protocols and existing reliable data transport protocols in wireless sensor networks, it was found that no existing protocol has all of the following characteristics:

- Full reliability (100% reliable data delivery) is provided unless there are unavoidable packet drops due to buffer overflow.
- Recovery from packet loss can be achieved with low system overhead and reduced communication cost and delay compared to conventional protocols.
- Lost packets can be recovered as quickly as possible, while at the same time not interfering with normal data transmission.
- A specified level of robustness can be provided.
- The protocol is robust to node failure and route changes.
- Fresh data has higher priority in the network and is able to be sent as soon as possible.
- The protocol is scalable and easy to implement.

It is the goal of this thesis to design, implement, and test such a protocol.

1.4 Contributions

This work proposes a new reliable data delivery protocol for general point-to-point data delivery (unicasting) in wireless sensor networks. The new reliable data transport protocol is designed that aims at providing 100% reliability when possible as well as minimizing overhead and network delay. The main contributions of this work are:

- A new negative acknowledgement (NACK) based reliable data transport protocol is proposed. The new protocol adopts a NACK-based hop-by-hop loss detection and recovery scheme using end-to-end sequence numbers.
- In order to solve the single/last packet problem in the NACK-based approach, a hybrid ACK/NACK scheme is proposed. Conventional NACK-based approaches are efficient in loss detection and recovery but they cannot provide reliable delivery of single/last packets. In the proposed new protocol, an ACK-based approach is used as a supplement to the NACK-based approach to solve the single/last packet problem.
- A new queue management scheme that gives priority to newer data is proposed. By introducing the idea of a ready bit and a newer packet first rule in the transmission queue, nodes can detect and recover lost packets in parallel with the normal data transmission process. Newer packets in the transmission queue don't have to experience any extra delay and can be transmitted as quickly as possible.
- A new protocol incorporating the above three schemes is implemented and tested. The new protocol is implemented and tested in a Crossbow MicaZ testbed under various system and protocol parameter settings. The performance of the new protocol is also studied and compared against four other protocols with different reliability schemes.

1.5 Thesis Organization

The rest of the thesis is organized as follows. Chapter 2 reviews background related to data transport protocols in wireless sensor networks. Chapter 3 presents the considerations and design of the proposed new protocol. Chapter 4 describes the performance evaluation methodology and experimental results from a protocol implementation. A summary of the contributions of this work and a discussion of possible future work are presented in Chapter 5.

CHAPTER 2
BACKGROUND

A larger number of wireless sensor network applications require reliable data delivery. However, due to the nature of sensor networks, designing a reliable data transport protocol faces many challenges, such as constrained energy, large number of nodes, data-centric networking, and small message size. This chapter presents an overview of general reliability issues in the data transport protocol for wireless sensor networks and discusses some recently proposed data transport protocols. Section 2.1 discusses the general issues in data transport protocol design in WSN. Section 2.2 and Section 2.3 provide overviews of reliable data delivery and congestion control in WSN, respectively. Section 2.4 presents a survey of existing data transport protocols for wireless sensor networks.

2.1 Data Transport Protocol Design in WSN

Because of the unique features of wireless sensor networks, the design of a reliable data transport protocol for WSN can be very challenging. Unlike the traditional TCP/IP network, each sensor node in a WSN has very limited power, bandwidth and storage space and has to cope with a lossy wireless channel. The reliable data transport protocols that are widely used in the internet such as TCP and UDP are not suitable for wireless sensor networks [32]. The following two sections discuss the design considerations of a reliable data transport protocol in wireless sensor networks and some commonly used performance evaluation metrics.

2.1.1 Design Considerations

In general, reliable data transport protocols for wireless sensor networks should consider a number of factors. First, the reliable data transport protocol should be able to

provide robustness to the network and be able to adapt to different scenarios, such as node failure and route changes. The initiation process of the transport protocol should be as simple and as quick as possible. For example, consider a remote monitoring wireless sensor network application, in which sensor nodes spend most of their lifetime in idle or sleep mode, but should be able to switch to transmitting mode and start the reliable data transport in a very short period of time when an event occurs in the network [39].

Second, since a WSN is an energy-constrained multi-hop network, a reliable data transport protocol should try to avoid any packet drop unless absolutely necessary. This is because data packets normally have to travel many hops before they reach their destinations. If a packet is dropped during the transmission, all the energy and bandwidth that have already been spent on the packet in the previous hops are completely wasted. However, there are cases where packet dropping is inevitable. Since sensor nodes have limited storage space, when the buffer is full of data packets and a new packet arrives, a data packet must be discarded.

Finally, fairness may be another consideration in the reliable data transport protocol design. As a data collecting network, most of the data flows are transmitted from sensor nodes to the sink. Such a multihop many-to-one routing structure can often result in unfairness in the network, in that the packets from nodes far away from the sink have a higher possibility to get lost during transmission than packets from closer nodes. Such unfairness for different nodes can cause problems in some applications and thus may need to be considered when designing a reliable data transport protocol [7].

2.1.2 Performance Evaluation Metrics

Because of the unique features of wireless sensor networks, the data transport protocol in a WSN should be able to mitigate network congestion, minimize overhead, reduce packet loss, and improve overall end-to-end reliability. In general, the metrics used to evaluate the performance of a WSN data transport protocol can be categorized as reliability, QoS (Quality of Service), and energy efficiency [17].

- Reliability: Reliability in wireless sensor networks can be examined from both the packet level and the event level. Packet level reliability refers to how many packets are successfully received at the final destination. An alternative way of measuring

packet level reliability is to calculate the overall end-to-end packet loss rate. The smaller the loss rate, the better the reliability of the network. Event level reliability refers to the delivery of certain data objects or events to the receiver. In this case, not all data packets need to be received by the receiver, rather, as long as a certain percentage of packets within a certain time period are delivered to the destination, the event level reliability is considered provided.

- QoS: In general, quality of service includes metrics such as bandwidth usage, network latency, real-time/in-order delivery and system throughput. QoS requirements can vary widely for different wireless sensor network applications. For example, in a real-time monitoring application, the delivery of data in a timely manner is more important than other metrics. However, in a high-rate application such as imaging or acoustic localization, where concurrently transmitting a large volume of data from different nodes is required, efficient bandwidth usage is more crucial.
- Energy efficiency: Since wireless sensor nodes are typically powered by battery and may be deployed in a remote rural area, the ability to collect data at a low energy cost is an important performance metric in wireless sensor networks. Energy efficiency can be examined by calculating the total energy spent in the network with a certain percentile of reliability. For example, with the STCP protocol, the energy cost for providing 100% reliability in a 100-node network for 100ms was 2.78 J, for 75% reliability was 1.06 J, and for 50% reliability was 0.77 J [16].

2.2 Reliable Data Delivery

Reliable data delivery is a critical issue in the application of wireless sensor networks. The requirement of reliability may vary from application to application. However, the fundamental issue of reliability is the same: how to detect packet loss and how to repair it. There are many design options for reliable data transport protocols in wireless sensor networks. For example, the protocol can run on an end-to-end or a hop-by-hop basis, the loss recovery scheme can be based on positive acknowledgment or negative acknowledgment, and the reliable data delivery can be provided from the direction of sensors-to-sink or the direction of sink-to-sensors. In Section 2.2.1, some of the general

issues in reliable data delivery protocol design are discussed. Section 2.2.2 presents several basic approaches of achieving reliability in wireless sensor networks from three different perspectives.

2.2.1 Discussion of General Issues in Reliable Data Delivery

In general, a reliable data transport protocol should cover the following dimensions:

- Communication type: Reliable protocols should provide reliable delivery of a single packet, blocks of packets or streams of packets [17]. Streams of packets are a continuous data streams. Periodic event monitoring is an example application type using streams of packets. Blocks of packets are segments of a complete data stream. A block of packets consists of a fixed number of data packets. Reliably delivering a single packet can be very important for queries or highly aggregated data, while delivery of blocks of data is necessary for many WSN applications such as remote network reprogramming. The cases of delivering a single packet and delivering blocks of packets can use very different underlying protocol mechanisms. The primary approaches for single packet delivery are ACK-based retransmission and transmission of multiple redundant packets. A wider variety of options exist for reliable delivery of blocks of data or streams of data. NACK-based approaches and multi-paths approaches are commonly used in such protocols.
- Reliability Requirement: Reliability requirements vary across different wireless sensor network applications. For sensors-to-sink delivery, the reliability requirement is either 100% guaranteed data delivery (or as close to this possible) or a percentage or probabilistic delivery requirement (for example 75% reliability). For sink-to-sensors delivery, the reliability requirement can be classified into four categories: 1) delivery to the entire network; 2) delivery to sensor nodes in a sub-region of a network (location based delivery); 3) delivery to the core members of the network that are able to cover the entire sensing field; and 4) delivery to sensor nodes with a probabilistic reliability requirement [7].
- Upstream and downstream delivery: In wireless sensor networks, it can be assumed that most communications are not between arbitrary peer nodes. As a data collecting network, the data flow in wireless sensor networks is normally from sensor nodes

towards a single sink/gateway node. Most of the research in WSN reliability is dedicated to sensor-to-sink transmission, such as in the protocols RMST [29], RBC [42] and RCRT [21]. However, in some scenarios, a reliable protocol for downstream communication is also important. For example, if a network consists of reprogrammable sensor nodes, the sink may want to send out certain control codes such as upgrade commands or new code images to the nodes. In the sink-to-sensors communication, since there is only a single sender (the sink), the data transmission usually uses broadcasting rather than unicasting. Reliable downstream protocols include PALER [20], PSFQ [32], HRS [19] and GARUDA [22].

2.2.2 Basic Approaches in Reliable Data Delivery

In this section, basic approaches to achieving reliability in wireless sensor networks are discussed from three different perspectives.

2.2.2.1 End-to-End vs Hop-by-Hop Error Recovery

In order to achieve reliability, the reliable data delivery protocol should be able to recover the lost data when errors happen. In traditional IP networks, the commonly used error recovery mechanism uses end-to-end acknowledgments, in which the final destination node is responsible to detect lost data and request retransmission. There are two critical challenges for the end-to-end recovery to be applied in wireless sensor networks. Firstly, in sensor networks, the connectivity between sensor nodes is neither stable nor predictable. The high latency and frequent disconnects generate significant overhead and delay and thus may seriously compromise the effectiveness of end-to-end recovery. Secondly, since sensor networks rely heavily on multihop forwarding to transmit information, while the end-to-end recovery mechanism only detects loss at the last node, the probability of successful reception at the destination node may become quite low. Wan *et al*. illustrate this problem by giving a simple example: assume that the packet error rate of a wireless channel is p, then the probability of successfully receiving a packet transmitted from n hops away is only $(1-p)^n$ [32]. These two problems indicate that end-to-end recovery is not an ideal choice for reliable data transport in wireless sensor networks.

Hop-by-hop error recovery is used in protocols such as PSFQ [32] and has become a widely-accepted recovery mechanism in sensor networks. The basic design idea of hop-by-hop error recovery is that the intermediate nodes, rather than just the final node, perform loss detection and recovery. To be specific, the whole multihop forwarding operation is divided into a series of single-hop processes. By ensuring reliable transmission between every two neighbor nodes in the transmission path, overall reliability can be achieved. The biggest advantages of hop-by-hop recovery are that recovery from packet loss can occur quickly, and progress made in early hops is not lost if a failure occurs in a later hop.

However, hop-by-hop recovery also has some shortcomings, which should be carefully considered when designing a hop-by-hop based protocol. The most obvious problem is the accumulated delay during multihop transmission. Typical hop-by-hop recovery schemes check for packet loss with every transmission on every hop, even when the network situation is good and no packet loss exists. This can generate significant overhead and unnecessary delay. Another problem of hop-by-hop recovery is that, in order to recover lost packets, intermediate nodes have to buffer all the incoming packets. This is not always desirable. On one hand, when the network condition is good and retransmission does not happen very often, buffering all the packets is a waste of resources. On the other hand, if some events suddenly happen after a long quiet time, a large amount of data can be generated and fill all the buffer space of the intermediate nodes in a short period. In this case, the effectiveness of hop-by-hop recovery is compromised [32].

2.2.2.2 ACK vs NACK

ACK and NACK are two commonly used receiver feedback mechanisms in multi-hop wireless networks. An ACK is the control packet sent by the receiver if it has successfully received the data packet from the sender. Normally, ACK-based loss recovery schemes are timer-driven. That is, if the sender doesn't receive the ACK from the receiver within a predefined period, the sender will consider the data packet to be lost during the transmission and will resend the previous packet. NACK-based loss recovery schemes work in a different way. If the receiver doesn't receive the data packet within a

given time, it will send back a NACK packet to the sender to request retransmission.

ACK-based approaches seem to be more reliable than NACK-based approaches since they verify the transmission of every single packet. However, ACK-based schemes suffer from two major drawbacks when used in sensor networks. The first problem is that, considering the limited bandwidth and energy of sensor nodes, the overhead of sending an ACK for every data packet may be unacceptable, especially when the size of each data packet is relatively small. The second problem is the well-known ACK implosion problem [32]. That is, when a node is broadcasting data packets in a dense network, the requirement of sending an ACK in response to the receipt of a packet for all the receivers may cause serious channel congestion and packet collisions. NACK-based is more effective than ACK-based and can be a better option for sensor networks because it only generates an extra packet when data loss occurs. However, when designing a NACK-based loss recovery scheme, several issues still need to be carefully considered. Similarly as with ACK-based schemes, there is a potential NACK implosion problem. When the network connectivity is poor and the sender is broadcasting to many receivers, the sender can be flooded with NACK packets. Retransmissions may lead to more serious congestion in the network, while ignoring errors can reduce overall network reliability. Another typical NACK problem is the loss of all data packets. In a NACK-based scheme, the receiver can detect and report packet loss only if it is aware of the incoming packet. Thus, a NACK-based scheme cannot handle the unique case where all packets in a communication are lost.

2.2.2.3 Sender Retransmission vs Forward Error Correction

Sender retransmission and the transmission of redundant data are the two basic ways of providing reliability for transport protocols. Automatic repeat request (ARQ) is the most commonly used sender retransmission method. In ARQ schemes, the sender will retransmit the packet if loss occurs and an acknowledgement (ACK) is not received prior to expiry of a retransmission timer [27].

Forward error correction (FEC), in contrast to ARQ, provides reliability by transmitting redundant packets in a proactive manner. Block (n, k) codes are the most

commonly used FEC codes. In (n, k) FEC [35], an additional n-k packets are added to each group of k source packets. The successfully receipt at the receiver of any k packets out of the n transmitted packets enable the reconstruction of the original k packets. There are many different FEC codes. The XOR code is a simple ($k+1$, k) FEC. Each transmission group only adds one parity packet, which is the bitwise XOR of all the source packets in the group. The XOR code is relatively simple to implement, but can only repair one packet loss in the group. RS codes [35] are block (n, k) FEC codes that have multiple parity packets. RS codes are more flexible and can provide better protection against losses. However, they can result in high processing costs and additional memory space requirements. Tornado codes provide an alternative to RS codes [35]. Tornado codes require a few more than k encoded packets to recover k source packets, but they have lower computational complexity and smaller reception overhead than RS codes.

Both ARQ and FEC are appealing approaches to achieving reliability. ARQ schemes are very effective and can always recover loss as long as the network is connected and there is sufficient node buffer space. However, if the error rate is high or link failure happens frequently, the cost of ARQ for loss detection and retransmission can be high. Using FEC can avoid the overhead generated by ARQ and prevent feedback implosion in large scale data transmission. However, FEC schemes should not be applied in congested networks. When the network is congested, adding redundant data will only aggravate the situation.

From the discussion so far, both ARQ and FEC methods have their advantages and disadvantages when applied in wireless sensor networks. A natural idea is to combine these two schemes together. For example, a low-overhead FEC code can be applied for transmission of data packets, while uncorrected errors are handled using ARQ. Wang *et al.* [35] propose a new reliability scheme for network reprogramming that uses a hybrid of ARQ and FEC. Use of FEC provides an abstraction of a better transmission medium and an ARQ scheme is responsible for the remaining loss recovery.

2.3 Congestion Control

Congestion can occur in wireless sensor networks due to several reasons: interference between concurrent data transmissions, the addition or removal of sensor nodes in the network, or bursts of messages because of the occurrence of some events [31] [41]. Congestion in the network can lead to two serious outcomes. As congestion spreads, buffer drops will increase quickly and become the dominant reason for packet loss. Significant delay can also be observed when congestion occurs. Another consequence of congestion is the growing expenditure of resources per packet. Fewer packets can be transmitted with the same amount of energy as before. Thus, alleviating congestion can be helpful in achieving reliable data delivery. The design and implementation of a congestion control protocol is challenging in the wireless sensor network domain due to the following reasons: firstly, the wireless channel itself is lossy and uncertain, which makes distributed data flow control a challenge; secondly, contention for the wireless channel can be observed at both the sender and receiver side; and finally, it is difficult to optimize channel utilization and fairness at the same time.

For sensor networks with a single sink node, mitigating congestion is mainly done by employing passive approaches. Rate control is the most commonly-used method [36]. When congestion is detected in the network, sensor nodes limit their reporting rate and thus give opportunity for congested nodes to drain their queue. Many papers have studied such rate control methods and focused on how to dynamically adjust the reporting rate in the context of various congestion situations. Another method that can be used to alleviate congestion is packet dropping. When the receiver node has already used up all the buffer space due to congestion, it clearly has to drop either the newly-arrived packet or an old one. In this case, evaluating the importance of different packets becomes important in that it can help a node to make better dropping decisions to avoid wasting resources.

2.3.1 Congestion Detection

Two fundamental methods have been proposed so far to detect congestion in sensor networks. Based on the observation that congestion can result in excessive queueing, the first method is to compare the instantaneous buffer occupancy with a certain watermark.

If the water mark is exceeded, a congestion state is diagnosed. This method is simple to implement. However, its accuracy is questionable, especially when packets are already lost on the channel. Another way to detect congestion is through channel sampling. As used in CODA [34], when a packet is waiting to be sent, the sensor node samples the state of the channel at a fixed interval. Based on the number of times the channel is sensed busy, a utilization factor can be calculated to deduce the congestion level of the network.

In wireless sensor networks, the sink is normally considered to have unlimited resources and able to have a more extensive view of the network behavior than a normal sensor node. Thus, in some protocols such as RCRT [21], the sink makes all the congestion detection and rate allocation decisions.

2.3.2 Congestion Notification

When network congestion is detected, the congestion notification information needs to be conveyed from the congested nodes to their neighbors or to the source nodes or destination nodes. The method for delivery of notification information should be carefully designed since sending new messages into an already congested network could only aggravate the situation. The congestion information can be sent in the form of a congestion notification (CN) bit in packet header or in a more comprehensive format that includes the congestion degree or allowable data rate. The congestion information can be sent in an explicit control message to notify the relevant nodes. It can also be sent in an implicit way by including control information in a regular data packet. For example, in ESRT [28], when congestion is detected, the sensor node sets a CN bit in the header of the packet being forward. By checking the header of an incoming packet, the receiver/sink can learn the congestion status of the network.

2.3.3 Rate Adjustment

A straightforward way of alleviating congestion is to simply stop sending packets into the network, or to send at a lower rate. The rate adjustment decision can be made by the congested nodes themselves, by a node outside the congested area (sink node), or by a predetermined policy. When a single CN bit is used to notify congestion, one option is for

nodes to adjust their sending rate according to an additive increase multiplicative decrease (AIMD) scheme as in RCRT [21]. On the other hand, if additional congestion information is provided such as congestion degree or allowable data rate, nodes can implement a more accurate rate adjustment as in CCF [5].

2.4 Existing Protocols

In general, most of the data transport protocols proposed for wireless sensor networks focus on either reliable data delivery or on congestion control. Only a few protocols such as RCRT [21] and STCP [16], deal with both issues. The protocols focusing on reliable data delivery can be further classified according to their assumptions regarding the data transmission direction. Sensors-to-sink protocols include RMST [29], RBC [42], DTSN [26] and Flush [18]. Sink-to-Sensors protocols include PSFQ [32], PALER [20], GARUDA [22] and HRS [19]. A classification of the existing data transport protocols discussed in this thesis is shown in Figure 2.1.

2.4.1 Protocols with Reliability Guarantee

Pump Slow Fetch Quickly (PSFQ) is a classic transport layer protocol proposed by Wan *et al*. [32]. In PSFQ, a source distributes data packets at a relatively slow pace to a network of sensor nodes. Whenever a packet is detected as being lost (due to out-of-order packet reception), PSFQ will fetch the missing data very aggressively (quickly) by sending an immediate NACK to the node's one-hop neighbors. In PSFQ, in-order reception is a very important requirement. Nodes only forward packets that are received without a gap in their sequence numbers. This requirement not only prevents the propagation of a loss event to a node's downstream neighbors but also helps to recover the lost packet very quickly since an immediate retransmission can be requested and established. PSFQ introduces the concept of localized recovery, which is designed to reduce recovery costs and network overhead by suppressing the redundant retransmission requests and the propagation of a loss event. The pushing mechanism and fetching mechanism in the PSFQ protocol is built in a tightly controlled timing manner. When a node receives a segment, if the segment is out-of-order, the node will prepare a NACK requesting the missing segments and will continuously send out NACKs, spacing

according to a parameter Tr until all missing segments are recovered. Otherwise, the node will schedule a forwarding event with a random delay between $Tmin$ and $Tmax$. The relationship between Tr and $Tmax$ is crucial in PSFQ, since the ratio $Tmax/Tr$ determines how aggressive the node is in trying to recover a missing segment. When a node receives a retransmission request, the node will first check its own cache to locate the requested segments and if found, schedules a resend with a random time between 1/4 Tr and 1/2 Tr. If the node overhears a response with the same missing segment from other neighbors before its own reply, the node will cancel the resend event in order to reduce contention as well as redundancy.

Reliable Multi-Segment Transport (RMST) is a NACK-based protocol that has primarily timer-driven loss detection and repair mechanisms [19]. RMST is designed for relatively long-lived data flows from source nodes to a sink node, although it could be applied to other contexts as well. RMST combines both transport layer and MAC layer mechanisms to achieve reliable data delivery. In RMST's cache mode, all intermediate nodes on a path as well as the sink maintain a cache that stores all segments being sent. When a node detects a missing segment, it creates a NACK packet that includes the missing segment's identifier and sends it back along the path to the source. When an intermediate node receives a NACK, if it has all of the missing segments listed in the NACK in its cache, it will forward them towards the sink and drops the NACK. If the node has only some or none of the missing segments, it locates and resends the missing segments it has (if any) and forwards the NACK again along the path to the source until either all missing segments are recovered or the NACK reaches the original source node. In RMST's non-cached mode, only the sink and the source node have the ability to maintain such a cache. Thus, when a missing segment is identified, the NACK travels from the sink all the way back to the source node and the source will resend the missing segment in a timely manner. The MAC layer design in RMST is important in that it not only provides hop level error recovery for the transport layer but it is also necessary for the discovery and maintenance of the route from source to sink. RMST is found to achieve good performance in networks with high connectivity and low error rate.

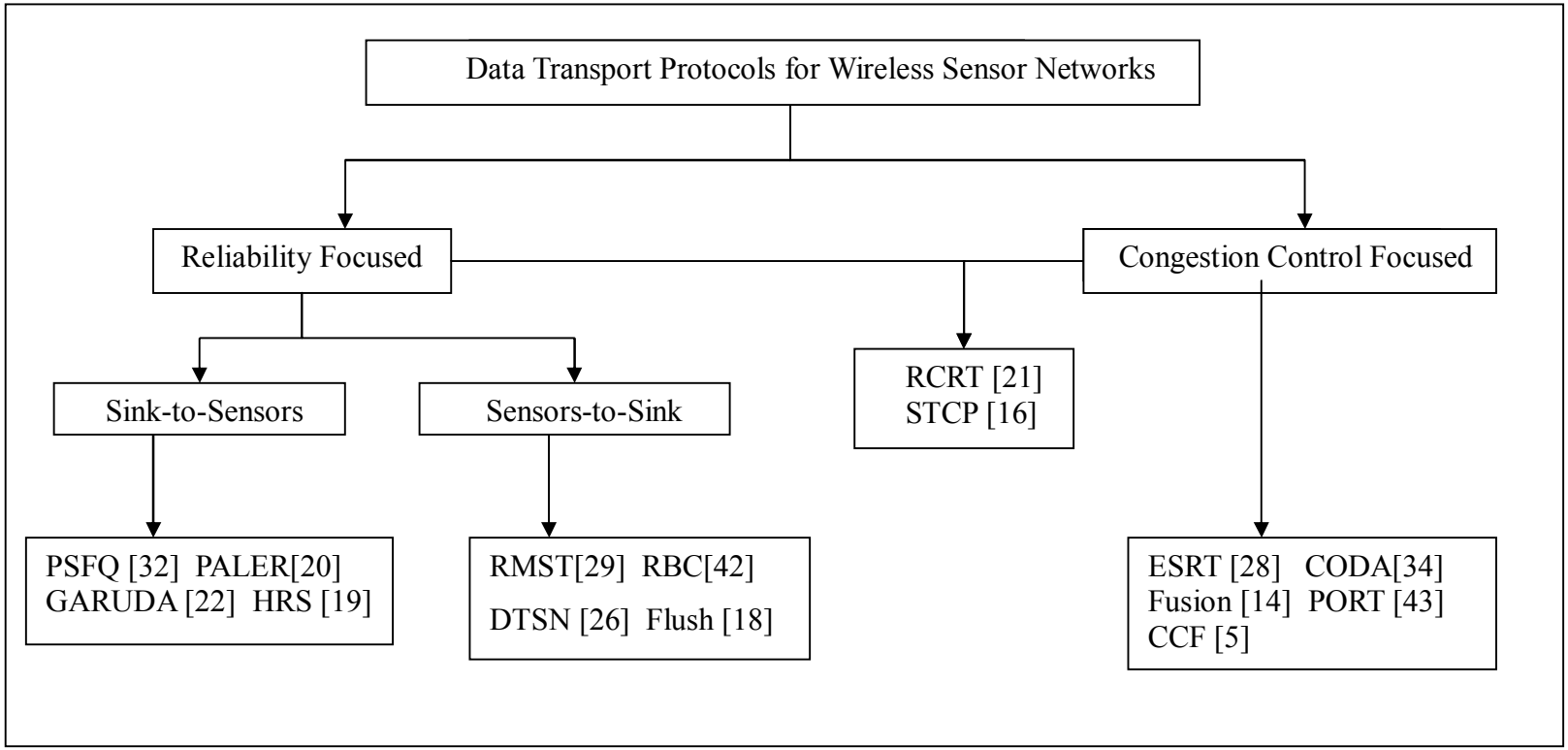


Figure 2.1 Classification of Existing Reliable Transport Protocols for Wireless Sensor Networks

GARUDA [22] is a reliable protocol designed for wireless sensor networks that focuses on reliably transferring blocks of data from the sink to the rest of the network. GARUDA uses a NACK-based approach similar to PSFQ but also incorporates a scheme that guarantees the reliable delivery of the first packet of a data stream. This scheme effectively solves the problem associated with NACK-based protocols that a receiver needs to receive at least one packet from a block of data in order to detect packet loss. The sensor network topology in GARUDA is constructed as an approximation to the minimum dominating set (MDS). Each sensor node in the network is classified as either a core member or a non-core member. The core member construction is done by a single packet flood. The core members of GARUDA act as the recovery center for downstream core members as well as non-core members and they each know at least one upstream core member. When there is no core member in the range of a non-core member, the non-core member can send out a request for a nearby core member candidate to become a core member. In GARUDA, an upstream core member includes in every forwarded packet a bitmap indicating the availability of its current segments into every forwarding packet. When a segment is missing, GARUDA implements a two-stage loss recovery method. A core member simply sends out a NACK to its upstream core node requesting retransmission of the missing segment. This recovery process is carried on in parallel with the default message-forwarding process in order to reduce network latency. A non-core node snoops the network and only requests packet retransmission from its associated core node when a complete bitmap is overheard from the core node. The reliable single/first packet delivery in GARUDA is done by a pulsing-based approach. The sink transmits a small series of short pulses as a signal before it initiates the transmission of a block of data. Upon reception of the pulse from the sink, nodes reply with the same pulse to indicate their awareness of the incoming packet. Sensor nodes can also use the pulse to request retransmission of the first/single packet if they don't receive it.

Miller *et al.* propose and analyze PALER, a reliable transport protocol for re-tasking and remote programming of wireless sensor networks [20]. PALER is built on the previously proposed PSFQ. PALER is motivated by a belief that the aggressive local recovery method used by PSFQ can generate significant packet contention and collisions.

Thus, PALER introduces a lazy error recovery scheme with a more aggressive push scheme to help to relieve the channel contention. The complicated recovery mechanism used in PSFQ is replaced with a single inclusive NACK scheme in PALER. Furthermore, by using local neighbor information and examining the local cache of received packets, PALER is also able to effectively detect and reduce redundant transmissions. PALER removes the in-order packet reception requirement as used in PSFQ. When a node receives a segment, if the segment has already been received and is still in the cache, the node increases the counter for it. If it is the first time the node receives this segment, the node schedules a forwarding event for this segment. The forwarding event will be cancelled if the counter for the segment reaches three before the segment is sent so as to reduce redundant transmissions. When a node receives the last segment of a data object, it broadcasts a NACK that contains a list of its missing segments to its one-hop neighbors. The broadcasting of the NACK is scheduled with a random delay period in order to reduce packet collisions as well as to give the node an opportunity to snoop rebroadcast segments in the network. If a segment is snooped before the NACK is sent out, this segment will be removed from the NACK. When a neighbor receives a NACK, it checks its cache to locate any of the segments mentioned in the NACK and schedules rebroadcasting of those segments.

Reliable Bursty Convergecast (RBC) is proposed by Zhang *et al.* to provide real-time and reliable data transport under conditions of high-volume bursty traffic [42]. RBC improves typical network efficiency by using a window-less block acknowledgement scheme to carefully schedule packet retransmission. Each sender in the network divides its packet queue into M+2 separate queues, indexed 0 through M+1 where M is the maximum number of retransmissions allowed at each hop. Packets in queue j have transmission priority over packets in queue j+1. Queue M+1 is used for free packet buffer. When the sender sends a packet to the receiver, the ID of the buffer holding the packet, as well as the ID of the buffer storing the packet to be sent next, are included with the data packet. When the receiver receives a packet from the sender, by comparing the buffer's ID with the expected buffer's ID piggybacked in the previous packet, the receiver can decide whether there is packet loss or not. There are two types of block feedback packets

in RBC. For a maximum number of packets that are successful received at the receiver without any missing packets in the middle, a block-ACK is generated. A block-ACK includes the sequence numbers of all successfully received packets in one block. In the case of missing packets, a block-NACK that records the sequence number of the expected packet and last received out-of-order packet is created and sent to the sender. Senders in RBC maintain a retransmission timer. The retransmission timer is set whenever a packet is sent. If no corresponding block-ACK is received or a block-NACK is received before the timer expires, a retransmission process is initiated at the sender. The packets that need retransmission are moved to a higher-ranked queue, and wait to be transmitted. RBC is evaluated using a real world experiment with 49 Mica 2 motes. The evaluation result shows that RBC can double the packet delivery ratio and reduce end-to-end delay compared with a commonly used stop-and-wait implicit-ACK scheme.

Wang *et al.* propose use of a supervised learning technique to improve reliability in wireless sensor networks [37]. Supervised learning is a particular case of machine learning, where both inputs and outputs are given in the training phase and nodes can automatically extract knowledge of readily-available features and the quantity of interest [28]. The inputs to the proposed supervised learning technique are system-level metrics such as buffer occupancies, the received signal strength and the channel load assessment, and outputs are performance metrics such as the throughput and the number of retransmission. The proposed technique consists of two phases: an offline learning phase and an online classification phase. Two case studies are presented in the paper to demonstrate the advantages of supervised learning. In the second case study, which is more related to the topic of this thesis, an extension to PSFQ called SHARP is presented, which is a situation-aware reliable transport protocol. SHARP uses the knowledge it learns from the offline learning phase to manage its storage space and control its caching policy. By using the online classification phase, the proposed approach can help individual sensor nodes to make informed reliability decisions.

Rocha *et al*. design a block oriented reliable transport protocol called DTSN that focuses on unicast communication in wireless sensor networks [26]. The basic loss recovery mechanism used in DTSN is Selective Repeat Automatic Repeat Request, which

employs both positive and negative acknowledgements. In DTSN, a session is identified by the tuple <source address, destination address, application identifier, session number> and is defined as a source/destination relationship. A randomly selected session number is used to distinguish different sessions between the same source and destination nodes. Within each session, all packets are given a unique end-to-end sequence number. The Acknowledgement Window (AW) is defined as the number of packets that the source sends before creating an Explicit Acknowledgement Request (EAR). During each transmission session, after the source sends a number of packets that equals the size of the Acknowledgement Window, it sends out an EAR packet to request feedback from the destination and also starts an EAR timer. The value of both the AW and the EAR timer are adjusted according to the individual application. The destination of the session prepares and sends out a feedback packet after receiving the EAR. If no feedback packet is received by the source before the EAR timer expires, the source will retransmit the EAR packet. If an ACK is received by the source, indicating there is no packet loss during the last session, the source will free up its output buffer and end the current session. Otherwise, if a NACK is received, the source will check the bitmap included in the NACK packet, identify the gap(s) in the sequence numbers and retransmit the lost packet(s). In DTSN, all intermediate nodes in the path between source and destination maintain an output buffer to store the forwarded packets.

Rahnavard *et al.* propose CRBcast [24], a two-phase broadcasting scheme which is built on probabilistic broadcasting and application layer rateless coding. CRBcast is a FEC based approach that has reliability and energy efficiency as its major considerations. CRBcast consists of two phases. In the first phase of the protocol, the source simply broadcasts the encoded packets. Nodes that receive enough encoded packets to recover the original packets are called complete nodes, while nodes that can not recover all of the original packets are called incomplete nodes. In the second phase of the protocol, complete nodes will collaborate with incomplete nodes to help them collect additional encoded packets and recover the original packets. Two types of handshake messages are used between complete and incomplete nodes: advertisement messages (ADV) and request messages (REQ). When a node becomes a complete node, it communicates its status to its neighbors by sending out an ADV. Any incomplete node replies to the ADV

by transmitting back a REQ which includes the required number of new packets for its completion, the ID of the complete node and certain flag bits. The complete node then responds to the REQ by sending the requested packets to the incomplete node. The advantage of CRBcast over a simple FEC scheme is that neighboring as well as downstream nodes work together to recover lost packets. Thus, the probability of successful reconstruction is increased. Two extensions of the CRBcast protocol, probabilistic advertising and multi-stage recovery, are also discussed in the paper.

Flush [18] is a receiver-based reliable bulk transfer protocol designed for multihop wireless sensor networks. Flush uses end-to-end selective negative acknowledgments to achieve end-to-end reliability, a dynamic rate control technique to maximize the usage of bandwidth and implicit snooping control messages to reduce system overhead. In Flush, there are four phases within a data transmission process: topology query, data transfer, acknowledgement and integrity check [18]. The receiver (sink) sends a topology query to request a data object as well as to calculate the Round Trip Time (RTT) to the source. The source then starts to send packets at the fastest rate possible without causing network congestion. Along the transmission route to the sink, Flush continuously calculates the usage of bandwidth and adjusts the transmission rate using the information snooped at each hop. The sink is responsible for tracking the received packets and detecting any packet loss. When the data transfer phase is completed, the sink sends out a NACK which contains the identities of any missing packets to the source for retransmission. When the NACK is received by the source, it resends the requested packets. When all packets are received at the sink, it starts to verify the integrity of the data. The sink discards the data object if it fails the integrity check; otherwise the sink keeps the received data object and requests the next one, if any. In order to reduce the sending time and increase the sending rate, Flush uses two basic rules in its rate control algorithm. Rule1 is that a node only sends packets when the downstream node is free. Rule 2 is that a node cannot send faster than its downstream node. Flush is tested with 79 Intel Mirage sensor nodes in a 48-hop network.

A summary of some of the existing protocols guaranteeing reliable data delivery is presented in Table 2.1.

Table 2.1 Summary of Existing Reliability Guaranteed Protocols

Features	Protocols							
	PSFQ [32]	RMST [29]	GARUDA [22]	PALER [20]	RBC [42]	DTSN [26]	STCP [16]	Flush [18]
Design Focus	Reliability, energy-efficiency and scalability	Reliability	Reliability and energy-efficiency	Reliability	Reliability	Reliability, energy-efficiency and scalability	Reliability and scalability	Reliability and time-efficiency
Direction	Sink-to-sensors	Sensors-to-sink	Sink-to-sensors	Sink-to-sensors	Sensors-to-sink	Sensors-to-sink	Sensors-to-sink	Sensors-to-sink
Loss Detection	Timer-based NACK	Selective NACK	NACK	Inclusive NACK	Implicit ACK	Selective ACK and NACK	ACK and NACK	Selective NACK
Loss Recovery	Hop-by-hop	End-to-end and hop-by-hop	Two-tier two stage loss recovery	Hop-by-hop	Hop-by-hop	Hop-by-hop	Hop-by-hop	End-to-end
Reliability	Packet reliability	Packet reliability	Packet reliability and cover sensing field reliability	Packet reliability	Packet reliability	Packet reliability	Packet reliability and probabilistic reliability	Packet reliability
Communication Type	Block of packets	Block of packets	Single packet and stream of packets	Block of packets	Block of packets	Block of packets	Block of packets and stream of packets	Block of packets
Unique Design	Pump slowly and fetch quickly	Cross-layer design (Transport and MAC layer)	Wait-for-First-Packet (WFP) pulse for reliable first packet delivery	Single inclusive NACK	Window-less queue management	Combined ACK and NACK design	Packet loss recovery and congestion control	NACK and rate control design
Evaluation	NS-2 simulation and Rene2 motes experiment	NS-2 simulation	NS-2 simulation	Jist/Swans simulation	49 Mica2 motes experiment	TOSSIM simulation	TOSSIM simulation	100 MicaZ motes experiment

2.4.2 Protocols with Congestion Control

Sankarasubramaniam *et al.* propose the Event-to-Sink Reliable Transport (ESRT) protocol [28]. ESRT keeps the network in the optimum condition (reliable without congestion) by dynamically adjusting the reporting rate of upstream nodes as well as controlling the downstream congestion level according to the current network state. ESRT classifies the network into the following five different states based on different reliability and congestion levels:

- State 1: No congestion and low reliability. This is a state with very low sensor node transmission rates and no congestion. The "reliability", as measured by the ability of the sink to detect events reliably, from reports received from the sensors, is low owing to the low sensor reporting rates. There is plenty of bandwidth not being used.
- State 2: No congestion and high reliability. In this state, no congestion is observed in the network. The reliability level exceeds the required level because of the high reporting rate of the sensor nodes. The network meets the reliability requirement but sensor nodes consume more energy than necessary.
- State 3: Congestion and high reliability. In this state, the network becomes congested because sensor nodes report more frequently than required. The network is able to maintain higher than required reliability due to the higher reporting rate but also experiences congestion.
- State 4: Congestion and low reliability. This is the worst possible scenario where the network experiences congestion while the reliability level is below the required reliability level owing to packet loss. ESRT will reduce the transmission rate aggressively to attempt to bring the network back to the optimal state.
- State 5: The optimal state. In this state, the reliability level matches the required reliability level with minimum energy consumption. This is the target state.

By monitoring the network for congestion signs and observing the rate of received packets for a period of time, the sink can determine the current state of the network. The congestion level is detected by measuring node buffer occupancies. Based on the detected congestion state, the sink will compute the new transmission rate to adjust the network to

the optimal state (state 5) and propagate this new rate to the network. ESRT assumes that in the non-congested state, a linear relationship exists between the transmission rate and the number of packets delivered at the sink per round. Under this assumption, the protocol should always converge to the optimal state, if this state is feasible to achieve.

As seen with ESRT, controlling and adjusting the sensor nodes' transmission rate is directly related to the congestion control issue in wireless sensor networks. The CODA protocol incorporates a new rate control framework [34]. CODA consists of three components: 1) a congestion detection mechanism based on observing the transmission queue size at intermediate nodes and the wireless channel load, 2) an explicit congestion notification method that uses a local back-pressure mechanism to signal nodes to reduce the forwarding rate, and 3) a centralized rate-control technique that allows the sink to regulate the multi-source rates. CODA adjusts the sending rate in a manner similar to AIMD [21].

Fusion incorporates three techniques to address the congestion issue in wireless sensor networks: hop-by-hop flow control, rate limiting and a prioritized MAC [14]. Hop-by-hop flow control consists of both congestion detection and congestion mitigation. Two commonly-used congestion detection methods are tested in Fusion: queue size monitoring and channel sampling. Congestion mitigation is done at the node level by observing the network congestion bit included in each data packet's header and adjusting the sender's forwarding rate according to the congestion level. If a routing path is unable to sustain the current traffic load, the hop-by-hop backpressure will propagate back to the source and allow the flow control mechanism to throttle the sampling interval. The rate-limiting technique is designed to allow sensor nodes to only send at the same rate as their children, thus nodes close to the sink can have a smaller chance of being flooded with packets when congestion happens. Fusion employs a prioritized MAC to aid congestion control. The length of each node's randomized MAC backoff is designed to be a function of its congestion state. When a node is congested, the backoff window is adjusted to only one-fourth of the size of a regular node's window. Thus, higher priority is given to a congested node by the prioritized MAC, allowing congestion notification (in the form of the network congestion bit in each data packet header) to propagate faster.

The Price-Oriented Reliable Transport Protocol (PORT) has a design that combines the ideas of multi-path routing, rate control and rate adaptation to avoid network congestion [43]. Based on the observation that different sources make different contributions to improving the sink's knowledge of events, PORT gives each node a price, which is defined as the energy consumed for each packet successfully delivered from the node to the sink, by all the nodes in the corresponding network path. The node price, together with link loss rates, is used to dynamically allocate the outgoing traffic to mitigate congestion. PORT also incorporates an optimal routing scheme for in-network nodes. An optimal route can be constructed based on the estimation of link loss rate on the source-to-sink path.

Paek *et al*. propose RCRT [21], a reliable rate-controlled transport protocol suitable for high-rate wireless sensor network applications. RCRT is designed to reliably transfer large amounts of sensor data from multiple sources to multiple sinks without incurring network congestion. RCRT uses end-to-end explicit NACKs and retransmissions to recover lost packets and implements a congestion detection and rate adjustment function in the sinks. In RCRT, end-to-end reliability is provided by a NACK-based loss recovery scheme. Each source node buffers every packet being sent and sinks track the end-to-end sequence number. Once a gap in the sequence numbers is detected, the missing sequences numbers are added to a missing packet list and the list will be sent at the end of the data flow in a NACK packet to the source nodes. Upon receiving a NACK from the sink, source nodes will initiate an immediate retransmission of the missing packets. Congestion detection in RCRT is done by monitoring the time to recover end-to-end loss at the sink. The sink maintains an out-of-order list, and if it takes more than four round trip times for the sink to recover a missing packet, the protocol decides the network is congested. Whenever congestion is detected, RCRT applies rate decrease steps according to an AIMD approach in the sinks and propagates the new transmission rate to the network. RCRT is one of the few protocols that provides a solution to both the reliability and congestion control issues.

A summary of some of the existing congestion control protocols is presented in Table 2.2.

Table 2.2 Summary of Existing Congestion Control Protocols

Features	Protocols					
	ESRT [28]	CODA [34]	Fusion [14]	PORT [43]	RCRT [21]	CCF [5]
Design Focus	Energy-efficiency and congested control for event-based WSN	Congestion control and energy-efficiency	Congestion control in spanning-tree topology	Congestion control and energy-efficiency	Congestion control for high-rate application	Reliability and scalability
Congestion Detection	Queue size	Queue size and channel status	Queue size	Link loss rate and node's price	Time to recover loss	Packet service time
Congestion Notification	Implicit	Explicit	Implicit	Explicit	Implicit	Implicit
Congestion Mitigation	AIMD-like end-to-end rate adjustment	AIMD-like end-to-end rate adjustment	Stop-and-start hop-by-hop rate adjustment	Multi-path routing	AIMD-like end-to-end rate adjustment	Exact hop-by-hop rate adjustment
Unique Design	Event-to-sink congestion control and sink-based congestion detection	Receiver-based congestion detection	A prioritized MAC design	Sink-based optimization approach and local optimal routing scheme	Sink-based congestion detection and mitigation approach	Packet loss recovery and congestion control
Evaluation	NS-2 simulation	NS-2 simulation and Rene2 motes experiment	55 Mica2 motes experiment	NS-2 simulation	40 Tmote motes experiment	NS-2 simulation and 10 Mica2dot motes experiment

CHAPTER 3

HOP-BY-HOP RELIABLE DATA DELIVERY

The main contribution of this thesis is the design and evaluation of a reliable data delivery protocol for wireless sensor networks. This protocol employs a hop-by-hop loss detection and recovery scheme. The goal of this protocol is to provide high reliability for general unicast communication with low system overhead and network delay. These goals are achieved by efficiently scheduling packet transport and through use of a new explicit NACK with reliable last/single packet delivery approach.

This chapter presents the design of the proposed protocol. Section 3.1 describes some design considerations in developing the protocol. Section 3.2 presents terminology and assumptions. The protocol's packet queue structure and explicit NACK approach are described in Sections 3.3 and 3.4 respectively. The detailed operation of the proposed protocol is presented in Section 3.5. Section 3.6 discusses some additional protocol features and design variations.

3.1 Design Considerations

Sensor networks applications often involve periodic data collection at a sink node. For such applications there is a steady rate of data packet transmission. In contrast to a conventional network, where all data is commonly given the same importance, in sensor networks, new data can be more valuable to the user. Considering a real-time monitoring application, the user is normally more concerned with the current status of the network and the new information contained in the fresh data, and therefore this data is more

valuable than old data. Meanwhile, when using a NACK-based scheme, older data is more likely to have been received by the receiver. Thus, in this study, it is assumed that if data packets must be dropped owing to a node buffer being full, newer data is more valuable than older data.

A sensor node, as limited by its own capacity, is forced to drop packets when its buffer is filled. When a large number of packets arrive in a short time or new packets generated by the node outnumber the available buffer spaces, packet loss is inevitable. One of the most important design concerns of this protocol is how to ensure that data packets are successfully delivered except when loss is unavoidable due to limited buffer space.

Due to the high link error rate of wireless sensor networks, hop-by-hop packet recovery is usually preferred over end-to-end packet recovery. Nevertheless, most of the existing hop-by-hop control mechanisms do not schedule packet transmissions so as to minimize delay. In some of the protocols, a sent packet is not removed from the head of the transmission queue until feedback has been received. As a result, newly arrived packets cannot be transmitted immediately and have to wait for the previous packet to be acknowledged. Significant delay can be observed in such protocols. In some other protocols, retransmissions of the missing packets are given higher priorities over the transmissions of newer packets. As a result, when retransmission occurs, data packets that are already stored in the transmission queue but haven't been transmitted may experience long queueing delays. Meanwhile, the transmission queue can be quickly filled up with new received packets and any new incoming packets may have to be dropped due to the buffer overflow. Therefore, in the proposed work, a new design is introduced to schedule the transmission process in a more efficient measure.

As discussed in Chapter 2, several ACK-based or NACK-based network layer hop-by-hop error recovery protocols have been proposed for wireless sensor networks. However, none of them can achieve 100% reliable delivery with low delay and

transmission cost. ACK-based approaches provide better reliability, but inevitably increase overall network delay and overhead. NACK-based approaches are more efficient in lost packet detection but cannot work properly in the presence of route changes or losing blocks of packets. Thus, a natural idea is to provide a comprehensive solution that has the advantages of both ACK and NACK techniques. In this thesis, a timer-based ACK approach is used to handle the last/single packet delivery problem while an explicit NACK method is used to detect and repair packet loss for regular data packets. Excepting for loss caused by buffer overflow, 100% reliable data delivery is provided.

3.2 Terminology and Assumptions

3.2.1 Topology and Link Layer Setup

The proposed protocol is expected to work for general topologies. Thus, for a single node, there could be multiple sources of incoming packets as well as multiple destinations of outgoing packets. Only unicast communication is considered in this work. Routing is outside the scope of the current work, and so when an intermediate node fails, the problem of how to build an alternative route is assumed to be handled by a separate protocol. A discussion of how the proposed protocol can accommodate node failure and route changes is presented in Section 3.6.

3.2.2 Protocol Terminology

- Source/Destination/Sender/Receiver: The source node for a packet is the node at which the packet was originally generated, while the destination node is the final destination of the packet. On each hop to the destination, the packet is transmitted by a sender node and received by a receiver node.
- Ready_Bit: If a packet is ready to be sent for the first time by the sender, or ready to be retransmitted, the Ready_Bit is set to 1. Otherwise, the Ready_Bit is set to 0. Only

a packet with Ready_Bit equal to 1 can be transmitted by the node.

- ACK/NACK_Bit: This bit indicates whether the sender requests ACK or NACK for the sent packet. The ACK/NACK_Bit is stored in the header of a data packet. When the receiver receives a new packet, it needs to read the value of the ACK/NACK_Bit. If the ACK/NACK_Bit is equal to 0, the receiver has to send an ACK for that packet; if the ACK/NACK_Bit is equal to 1, the receiver does not need to return an ACK.
- Packet_ID: This term is used to refer to the combination of the source and destination node IDs (all sensor nodes are assumed to have a unique ID) and the end-to-end sequence number (assumed to be unique for all packets between a particular source/destination pair). Since the Packet_ID is globally unique, it can uniquely indentify data packets in the network.
- Last_ID: This field in the packet header is used to store the Packet_ID of the last new (not a retransmission) packet sent by the sender to the receiver. The contents of this field are determined when the packet is transmitted for the first time by the sender, and are not changed if packet transmission is unsuccessful and the packet must be retransmitted. When the sensor state has been initialized (or re- initialized) and there is no state information regarding such a packet, the Last_ID field is set to be NULL.
- R[S]: A state variable maintained by the receiver for each sender. The state variable R[S] is created to store the Packet_ID of the newest in-order data packet that has been received from the sender. The value of R[S] is updated after a new data packet is accepted by the receiver. A detailed description of how R[S] is used to detect missing packets is presented in Section 3.4.
- SKIP: The SKIP field is a single bit indicating the availability of the requested data packet at the sender. When the receiver requests a retransmission of a missing packet and the packet has been dropped from the queue, the sender sends the oldest available data packet in its queue that is destined for that receiver and sets its SKIP field to 1. Upon receiving a data packet with SKIP field equal to 1, the receiver realizes that the

requested missing packet is no longer available at the sender and so it takes the received packet as an in-order packet.

3.2.3 Packet Format

In the proposed work, there are two different types of packets: data packet and feedback packet. A data packet is a packet containing sensor readings or other data. All the data packets share the same header format. The header of a data packet contains four important fields: Packet_ID, ACK/NACK_Bit, Last_ID and SKIP.

There are two types of feedback packet in this work: ACK packet and NACK packet. ACK and NACK share the same packet format. They both have a one bit feedback type field and a four byte Packet_ID field, giving the ID of the newest in-order packet successfully received by the receiver. These two packets can be distinguished by using their feedback type bit: 0 refers to NACK packet, 1 refers to ACK packet.

3.3 Transmission queue Management

The common method to recover lost packets in a hop-by-hop recovery scheme is through retransmission. However, most current retransmission schemes may yield either excess redundant traffic or excess delay. As one goal of the presented work is to transmit new data packets as quickly as possible, new transmission queue management policies are designed for better scheduling retransmissions.

3.3.1 Enqueue and Transmission Policies

Each individual node has to maintain a transmission queue structure, whose responsibility is to temporarily store packets and manage the transmission. Every packet in the queue has a Ready_Bit to indicate its status as introduced in Section 3.2.2. If the Ready_Bit is 1, then the packet is ready to be sent; if the Ready_Bit is 0, it means the packet has been sent already and does not need to be resent at this time. This Ready_Bit

can be changed when a NACK is received or the packet is transmitted.

When a sender gets a new packet, either generated by the node itself or forwarded by an upstream neighbor, the Ready_Bit of the packet is set to 1 and the packet is stored at the tail of the queue. Whenever the sender is ready to begin a new transmission, it checks the Ready_Bit of the packets in the queue in the order from the head of the queue to the tail. The first packet with Ready_Bit equal to 1 will be sent and its Ready_Bit will be set to 0. After sending a packet and waiting a random delay period (so as to provide some spacing between transmissions), the node checks the Ready_Bit of the packets in the queue again and sends the next packet with Ready_Bit equal to 1.

In a conventional ACK-based approach, after a data packet has been transmitted, it stays at the head of the queue and no other packet can be transmitted until feedback is received from the receiver. A direct consequence of this approach is that significant delay can be incurred if the sender doesn't get the feedback as expected, due to network congestion or link error. As a result, packets newer than the sent packet have to wait in the queue and cannot be transmitted promptly. This send-and-wait strategy may work fine when network connectivity is good. However, it may cause significant delay in poor network conditions.

In the proposed protocol, by taking advantage of this transmission queue design, a source node can transmit multiple new data packets consecutively without waiting for feedback. For example, suppose that a packet X is transmitted. Its Ready_Bit is then set to 0. A data packet newer than packet X with Ready_Bit equal to 1 can then be transmitted. At the same time, the sender can retransmit any missing packet in parallel with the regular data packet transmitting process. For example, when a node receives a NACK for a packet Y, it locates the missing packet in its transmission queue if present and sets its Ready_Bit back to 1. Any packet older than packet Y that is destined for the same receiver must have been received and is thus removed from the queue, as described in Section 3.3.2. If there is no older packet destined to some other receivers with

Ready_Bit equal to 1, the retransmission of packet Y can be started immediately.

3.3.2 Dequeue Policy

Since sensor nodes can receive and generate data packets constantly, while their storage capacity is quite limited due to size, cost and power limitations, an appropriate dequeue policy is necessary in order to manage the buffer space more efficiently. In the proposed policy, packets are dequeued in the following scenarios:

- If the transmission queue reaches its maximum capacity and a new packet is received, the node discards the packet at the head of the queue and makes room for the newer packet. Since all packets in the queue move in the sequence from tail to head, the packet at the head of the queue has been in queue the longest and can therefore be considered to be the most likely to have been successfully received by its respective receiver.
- If an ACK or NACK is received, the Packet_ID is read from the ACK and compared with the IDs of packets in the transmission queue. Since the Packet_ID in the ACK or NACK gives the latest in-order packet received by the receiver (see Section 3.5), the packet X with matching Packet_ID (if still present in the queue) as well as all packets older than packet X that were sent to that receiver and are still present in the queue can be dequeued.

3.3.3 Summary

The advantages of the proposed transmission queue management policy are two-fold. On the one hand, a packet will not be dequeued before it is known it has been successfully received, or, if necessary, when it is the oldest packet in the queue maximizing the opportunity to re-send lost packets. On the other hand, the detection of packet loss does not interfere with normal packet transmission, and thus delay can be significantly reduced while still ensuring a high level of reliability.

3.4 Explicit NACK with Reliable Last/Single Packet Delivery

The crucial issue in providing reliable data delivery is how to detect and repair packet losses. In this work, in order to overcome the problems with conventional ACK and NACK protocols as mentioned in the previous chapter, a hybrid of a NACK-based technique and explicit positive acknowledgement is employed.

The advantage of a NACK-based approach is obvious: it is effective in detecting packet losses and also efficient in recovering from them. However, a typical problem with a NACK-based approach concerns the fact that the receiver has to be aware of the incoming packet. Otherwise, the receiver cannot send out a NACK to request retransmission. In the proposed work, every outgoing packet includes the Packet_ID of the last packet sent by the same sender. Thus, by examining the Packet_ID of the packet most recently received from the same sender and the Packet_ID included in the current received packet, the receiver can detect if there are any missing packets between the two receptions. A "send-and-wait" ACK approach is used to ensure the successful transmission of the last/single packet, and an explicit NACK approach for the rest of the packets. The following detailed description of this approach considers a single sender/receiver pair.

As introduced in Section 3.2, every data packet includes an ACK/NACK_Bit in the packet header, which indicates whether or not this packet is the last packet in a data stream. When the sender prepares the data packet, if it is known that there will be another packet that will be sent to the same receiver within some reasonable time period, it sets the ACK/NACK_Bit to 1; otherwise the sender sets it to be 0. In the latter case, the sender sends the packet to the receiver and starts a retransmission timer. Upon receiving a packet from the sender, the receiver first examines its ACK/NACK_Bit. If the ACK/NACK_Bit is 0, the receiver learns that this packet is the last packet it will receive from the sender. Thus, it creates an ACK packet and sends it back to the sender. In this

case, the sender keeps re-sending the last packet until the reception of the packet is confirmed by the receiver. Otherwise, the receiver only uses the explicit NACK approach as described below.

When a sender S sends a packet Y to the receiver, the Packet_ID of the last packet transmitted by the sender to that receiver is included in the Last_ID field of packet Y's header. The receiver maintains a local state variable R[S], which is used to store the Packet_ID of the last in-order packet received from sender S. After receiving packet X, the receiver stores the Packet_ID of packet X in R[S]. When a new packet arrives, by comparing the Packet_ID in its Last_ID field with R[S], the receiver can learn whether or not this is the next in-order packet. If the Packet_ID in the Last_ID field of the new packet matches with R[S], then the incoming packet is indeed the next in-order packet, and R[S] is updated to the ID of packet Y. If the packet needs to be transmitted another hop, the receiver inserts packet Y into the transmission queue. If, on the other hand, the Last_ID field does not match R[S], the received packet is not the next in-order packet, whose transmission must have been unsuccessful. The receiver drops the packet, creates a NACK packet containing R[S] and immediately sends it back to the sender requesting retransmission of all packets newer than that with ID of R[S].

Note that in the explicit NACK approach, the receiver only accepts in-order packets and drops all out-of-order packets. This policy is designed to ensure that packet delivery is 100% reliable, except possibly when a node's buffer reaches its capacity. Specifically, a receiver continually requests a missing packet until it has been successfully received, or such reception is no longer possible, as described in Section 3.5. However, a possible performance enhancement can be achieved by buffering packets that are received out of order, so as to eliminate the need for the sender to retransmit such packets. A protocol variant using out-of-order buffering is proposed and discussed in Section 3.6.5.

3.5 Protocol Operation

In this section, a detailed description of the operation of the proposed protocol is presented. The protocol operation is described for a general scenario.

3.5.1 Sender Operation

- Before the sender sends any packet, it first examines the Ready_Bit of the packets in the queue. The sender chooses the first (oldest) packet it encounters in the queue with the Ready_Bit equal to 1 as the packet to be sent. If the packet is being transmitted for the first time, and the sender knows that it will be sending another new packet to the same receiver soon, it sets the ACK/NACK_Bit in the packet header to 1; otherwise, it sets this bit to 0. In the latter case, the sender will not send any new data packet until it gets an ACK back.
- When sending a packet that is being transmitted for the first time, the sender puts the Packet_ID of the packet that it has most recently sent to the same receiver (excluding any retransmissions) into the Last_ID field in the packet header. If there is no such previous packet or if the sender has no memory of its ID (e.g., the sender has failed and lost data at least once since the previous packet was sent), it fills this field with NULL.
- When the sender sends a packet (either a new packet or a retransmission), with the ACK/NACK_Bit set to 0, it initiates a retransmission timer. If no acknowledgment is received for the packet before the timer expires, the sender sets the ready bit of the packet back to 1, which will cause the packet to be retransmitted. Before the retransmission timer is fired, the sender cannot transmit any newer packet to this receiver unless an acknowledgment is received. Whenever the sender finishes sending a packet, it sets the Ready_Bit of the sent packet to 0.
- When the sender receives an ACK for a packet X, the sender cancels the

retransmission timer. According to the dequeue policy described in Section 3.3, the sender can safely remove the packet X as well as all older packets that were sent to that receiver from the queue and move on to send the next available packet.

- When the sender receives a NACK containing the ID of packet Y, the sender sets the Ready_Bit of all enqueued packets newer than packet Y that were sent to the same receiver (if any) to 1 and dequeues packet Y (if found in the queue) and all older enqueued packets (if any) that were sent to that receiver. The sender has to resend any packets sent to that receiver that are newer than packet Y and that are found in the queue because the receiver does not accept out-of-order packets. Note that if packet Y is not found in the queue, any packets defined for the same receiver that are in the queue must be newer than packet Y. In this case, the first of the packets transmitted to the receiver (or the first new packet if no packets for that receiver were found in the queue) has its SKIP field set to 1 to indicate that no packets between packet Y and this packet are available.

3.5.2 Receiver Operation

- When a packet with the ACK/NACK_Bit set to 1 is received and the state variable R[S] matches the Last_ID field in the packet header, the receiver accepts the packet, and no acknowledgment is required. The receiver sets R[S] to the Packet_ID of the received packet.
- When a packet with the ACK/NACK_Bit set to 0 is received and the state variable R[S] matches the Last_ID field in the packet header, the receiver accepts the packet and transmits an ACK containing the Packet_ID of packet X back to the sender. The receiver sets R[S] to the Packet_ID of the received packet.
- When a packet with the SKIP field set to 1 is received, the receiver accepts the packet and assigns the Packet_ID of the received packet as the new value of R[S]. The receiver may have missed one or more packets from that sender, but if so, then

packets were dropped from the sender's queue and cannot be recovered.

- When a packet is received with a Last_ID field that does not match R[S], and the SKIP field is not set to 1, the receiver must have missed one or more packets from that sender. The received packet is dropped by the receiver and a NACK with ID field set to its R[S] is sent back to the sender.

3.6 Additional Features and Design Variations

In this section, some other details of the proposed protocol are discussed. Those details are worth mentioning because they explain how the protocol works in different and complex scenarios, such as nodes failure and route changes, variable reliability requirements and how to eliminate duplicated data packets. In Section 3.6.4, a variation of the proposed protocol is also discussed, which further improves the performance of the original protocol in a high loss rate environment by caching out-of-order packets at the intermediate nodes.

3.6.1 Variable Reliability

As discussed in Section 2.1.2, different applications in wireless sensor networks may have different reliability requirements. The data transport protocol for a WSN should be able to adapt the quality of service and type of services to be provided to the application requirements. For example, in an event-monitoring application, it may be possible to recognize an event by receiving only 80% of the data packets reporting the event. It may also be necessary to maximize the life time of the network. By slightly modifying the loss recovery scheme, the proposed protocol can easily adapt to the above requirement. Each node in the network updates a measure of link-level reliability between the node itself and its upstream node whenever it receives a new data packet. The link-level reliability is defined as the number of received in-order packets divided by the total number of transmitted packets which includes packets not successfully received. Each node in the

network also maintains a required reliability level, which may be predefined or updated by the sink during data collection. When a node receives an out-of-order packet, if the calculated average reliability level is above the required reliability level, the node will not create the NACK packet and request retransmission of the missing packet. By skipping the creation of a NACK packet and so avoiding the retransmission of an unnecessary data packet, the node is able to maintain the required reliability level while saving energy and bandwidth which can in turn increase the lifetime of the network.

3.6.2 Route Changes or Node Failure

Node failure or severe network congestion may result in route changes [15]. Adding new nodes to an existing sensor network may also cause such changes. How to detect node failure, coordinate newly added nodes and construct a new route depends on the underlying routing protocols and is outside the scope of this paper. However, a well-designed reliable transport protocol should be able to effectively handle the case of route change or node failure and perform robustly. Because the proposed protocol uses hop-by-hop loss detection and recovery and can initiate the data transmission process without any additional information exchange, a newly joined node (either a newly added node or a node just available in the new route) can be integrated into a route without any previous knowledge of the network.

Consider the example shown in Figure 3.1, where the sender is forwarding or sourcing a stream of packets and the new node is a node that has just joined the network, which either replaces a failed node or is part of a new route. Assuming that the sender has five packets available in its queue (from packet 3 to packet 7) and has already sent out packet 3 in a previous transmission, the sender continues to send packet 4 as shown in Figure 3.1(a). When the new node receives data packet 4, since the Last_ID field does not match the local variable R[S] (which is NULL), the new node considers this packet an out-of-order packet and drops it. Since the transmission queue of the new node is empty

and there is no record of previously received packets, the new node sends back a NACK with NULL in the Packet_ID field as shown in Figure 3.1(b). Upon receiving the NACK, the sender checks its queue and finds no packet with Packet_ID matching with NULL. Thus, the sender resets all the packets' Ready_Bit to 1 and resends the first packet available (which is packet 3) in its queue with SKIP field set to 1. The new node accepts the packet 3 as shown in Figure 3.1(c) because of the SKIP field, and updates its state variable R[S] to the Packet_ID of packet 3. The new node successfully joins the network within two sending rounds.

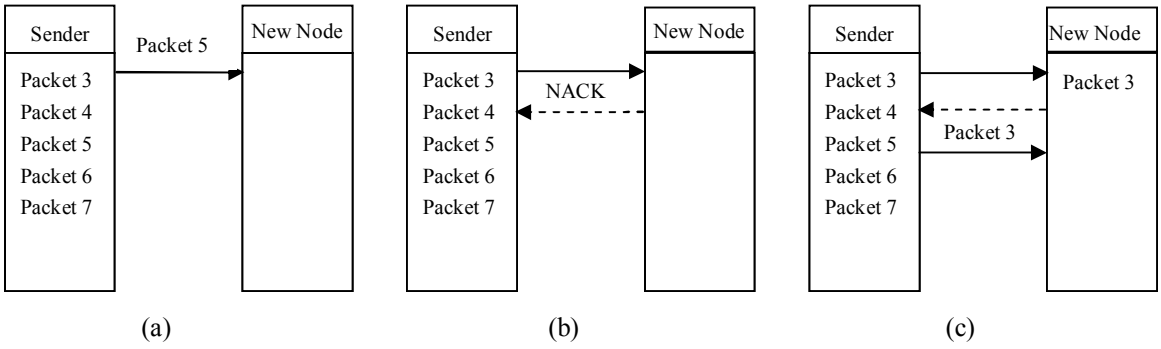


Figure 3.1 Example of Route Changes or Node Failure

3.6.3 Data Redundancy

Route changes and node failure can not only result in packet loss, but also the possibility of transmitting redundant data. As described in the previous section, when a new route is established or a new node joins the network, the sender needs to resend every packet available in its buffer to the new next hop node in case packet loss has occurred. However, some of these packets may have already been successfully transmitted to the destination nodes through the original path or other old nodes before the new node joins the path. Since all data packets will eventually be sent to destination nodes and the Packet_ID of each packet used in this protocol is globally unique, the data redundancy check can be done at destination nodes by comparing the Packet_ID of the

received packets.

3.6.4 New Protocol with Out-of-Order Buffering

In the hop-by-hop reliable data delivery protocol presented in the previous sections, the receiver accepts only in-order data packets and is forced to drop all out-of-order packets. As a consequence, when a NACK is received, the sender not only needs to resend the first missing packet but also all newer packets including the out-of-order packet. In an environment of high link error rate and poor network connection, this strategy can be a waste of resources. With a closer look at the sender operation, one can notice that the sender always sends data packets in a first-in-first-out (FIFO) basis. In other words, the sending sequence at the sender is consistent with the data packet arrival sequence at the sender's transmission queue, unless a retransmission timer expires or a NACK is received. Out-of-order packets the receiver receives are packets newer than the receiver's expected next new packet. Therefore, if out-of-order packets can be stored temporarily, at least one data packet retransmission can be saved by recovering the missing packet locally from the buffer. Therefore, a variant of the new protocol with out-of-order buffering is developed.

In the modified protocol, besides the transmission queue, each node also maintains a separate out-of-order buffer that can temporarily store a single out-of-order packet. If this buffer is already occupied when the node receives another out-of-order packet, the old packet in the buffer will be replaced with the new packet. The receiver still creates a NACK packet and sends it back to the sender for retransmission when it receives an out-of-order data packet. However, the receiver stores this packet into its out-of-order buffer, rather than immediately discarding it as in the original protocol. If a data packet with Last_ID matching R[S] is received next, the receiver updates its variable R[S] to the Packet_ID of the new packet and stores this packet in the queue (if it needs to be forwarded on). The receiver in the modified protocol also needs to examine the Last_ID

field of the packet in the out-of-order buffer. If the Last_ID field of the packet in the out-of-order buffer matches the new R[S], this packet is the next new packet from the sender. The receiver updates the state variable R[S] again to be the Packet_ID of the packet in the out-of-order buffer and moves this packet to the transmission queue (if it needs to be forwarded on).

When a data packet with SKIP field equal to 1 is received, there are several cases that need to be considered. If the out-of-order buffer is filled and the Packet_ID of the new packet matches with the Last_ID of the packet in the out-of-order buffer, the receiver saves the new packet and updates the state variable R[S] to be the Packet_ID of the packet in the out-of-order buffer and moves this packet to the transmission queue (if it needs to be forwarded on). If, however, the Packet_ID of the new packet doesn't match the Last_ID of the packet in the out-of-order buffer, the receiver needs to compare the Last_ID of the new packet and the Packet_ID of the buffered packet. If they match, the receiver moves the packet in the out-of-order buffer to the transmission queue and updates the state variable R[S] to be the Packet_ID of the new packet and saves the new packet in the transmission queue (if it needs to be forwarded on). In all other cases (e.g. the out-of-buffer is not filled or the received new packet is the same packet as in the out-of-order buffer), the receiver saves the new packet in the transmission queue and updates the state variable R[S] to be the Packet_ID of the new packet.

Because only an out-of-order data packet can trigger a NACK packet, when a NACK is received, the sender can assume that a packet it sent after the missing packet was received by the receiver and is stored in its out-of-order buffer. Thus, rather than retransmitting all newer packets in the queue as in the original protocol, the sender in the modified protocol only retransmits the oldest packet in its queue following the packet whose Packet_ID is given in the NACK. The example in Figure 3.2 is used to illustrate the mechanism of this modified protocol.

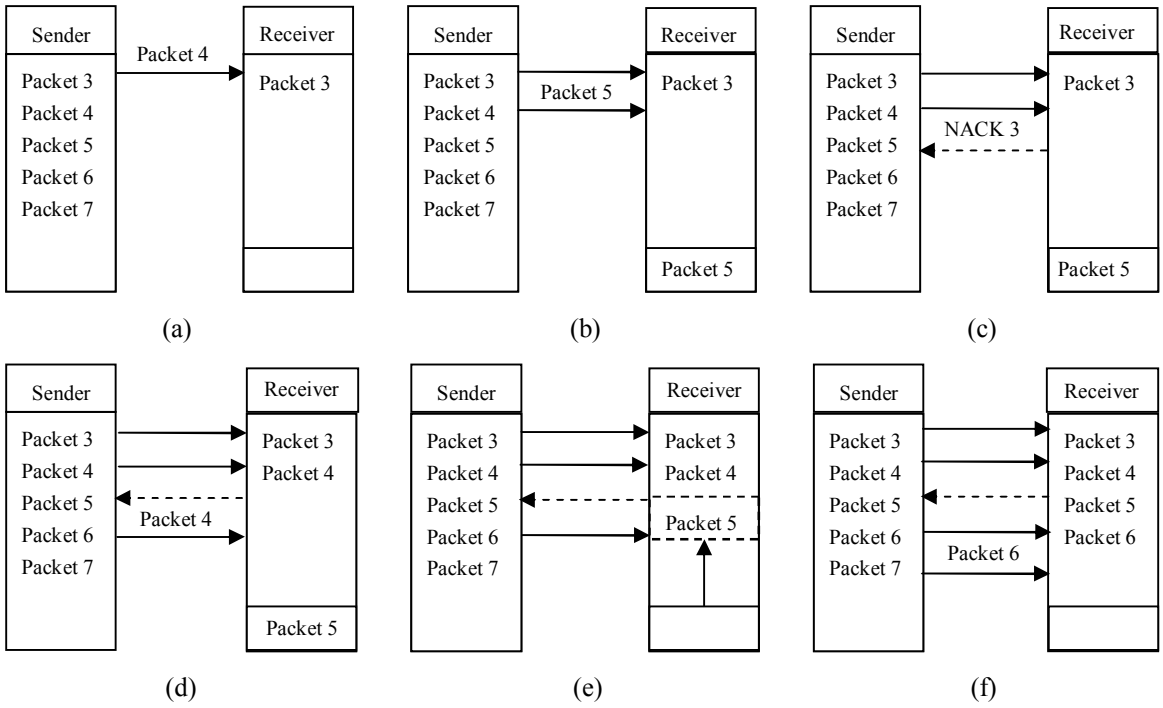


Figure 3.2 Example of New Protocol with Out-of-Order Buffering

In Figure 3.2, the left and right rectangles in each diagram represent the sender and the receiver, respectively. The receiver maintains a transmission queue, as well as an out-of-order buffer as shown in the bottom of the receiver rectangle. The solid line from the sender rectangle to the receiver rectangle represents the sending of a data packet, while a dashed line from the receiver rectangle to the sender rectangle represents the sending of a NACK. Assume that initially the last packet sent from the sender is packet 3 which has been successfully received by the receiver as shown in Figure 3.2(a). The transmission of packet 4 is lost but packet 5 is successfully received (Figure 3.2(b)). Since packet 5 is an out-of-order packet, the receiver puts it in the out-of-order buffer and sends back a NACK that containing the Packet_ID of packet 3 (Figure 3.2(c)). In Figure 3.2(d), after receiving the NACK, the sender locates the next in-order packet after packet 3, which is packet 4, and resends it. When the receiver gets the retransmission of packet 4, the receiver stores it in the transmission queue for forwarding since the Last_ID field in packet 4 (which specifies packet 3) is the same as R[S]. After receiving packet 4, the

receiver checks its out-of-order buffer and finds that the Last_ID in packet 5's header matches with the Packet_ID of packet 4. Thus, the receiver moves packet 5 to the transmission queue as shown in Figure3.2 (e) and updates R[S] to the Packet_ID of packet 5. The sender doesn't need to retransmit packet 5, and instead can transmit a new packet as shown in Figure3.2 (f).

CHAPTER 4

PERFORMANCE EVALUATION

This chapter presents a performance study of the proposed hop-by-hop reliable data delivery protocol based on a testbed implementation. Experimentation is chosen as opposed to simulation in order to get more realistic results. The performance of the protocol presented in Chapter 3 is evaluated under various network conditions. An overview of the performance study is described in Section 4.1. Section 4.2 describes the testbed implementation used to evaluate performance. Section 4.3 presents the methodology of the experimentation including performance metrics, the evaluated protocols, experiment parameters, and the experimental design. The experimental results are analyzed and discussed in Sections 4.4 through 4.6.

4.1 Performance Evaluation Overview

The performance evaluation study described in this chapter aims at demonstrating the performance of the proposed hop-by-hop reliable data delivery protocol. By employing techniques such as reliable last/single packet delivery and implicit NACK approaches, the new protocol is expected to have better performance on both overall network delay and reliability than conventional protocols. The experimentation on CrossBow Technology's MicaZ sensor nodes is described in this chapter and the following questions are considered in the experimental study:

- How is the overall performance of the new protocol in terms of link-to-link and end-to-end reliability?

- How is the performance of the new protocol impacted by various system and protocol parameters?
- Under what conditions will the performance of the new protocol be compromised?
- How does the performance of the new protocol compare with the performance of protocols with different loss recovery and detection schemes such as an ACK-based approach and a timer-based NACK approach?

4.2 Testbed

4.2.1 Software Implementation

The proposed protocol is implemented in the network embedded systems C (nesC) programming language and the TinyOS operating system [30]. nesC is a component-based event-driven programming language based on the C programming language. TinyOS is an open source component-based operating environment written in nesC and is optimized and designed for embedded systems such as wireless sensor networks. TinyOS consists of a set of software components and interfaces. The components in TinyOS are connected with each other by interfaces and provide common abstractions including routing, storage and communication [30]. A TinyOS application normally consists of one or more components. There are two types of components in nesC: *modules* and *configurations*. A module contains the implementation of an algorithm or a model. A configuration consists of the wiring of the components used in a module and describes the way components are connected by interfaces. All applications in TinyOS require a configuration file but not necessarily a module.

The implementations of all tested protocols in this thesis run on the Crossbow MicaZ platform. Each MicaZ mote has an ATMEL 7.37 MHz ATMega128L, low power 8-bit micro-controller with 128 KB of program memory, 512 KB measurement serial flash data memory, and 4 KB EEPROM [2]. The MicaZ mote uses Chipcon CC2420, a single-chip

IEEE 802.15.4 compliant radio frequency transceiver operating at 2.4 GHz, and is capable of transmitting at 250 kbps.

4.2.2 System Configuration

The testbed of the experimentation consists of up to 9 MicaZ motes (exclude interference nodes used in Section 4.4.4), depends on different experiments. In each experiment, one mote acts as the sink and is connected through a CrossBow MIB600 programming board to a computer. The sink is responsible for receiving data packets and logging information as well as broadcasting control messages. Other sensor nodes are programmed with the tested protocols. The computer is used to program/reprogram the MicaZ motes as well as receive and analyze data/log after the experiments. The default CSMA/CA (provided in TinyOS) is used as the MAC layer protocol for all MicaZ motes in the experiments. The new protocol is independent of the underlying network topology and routing protocol. The study in this work applies to other cases where different topology or routing protocol is used.

For experiments conducted in this thesis, all sensor nodes are deployed in a single line topology and the distance between two neighbor sensor nodes is 3 feet. The MicaZ mote's radio power level is set to -3dBm and the transmission range of the resulting network is just over 1 hop. All the nodes in the experiment are time-synchronized prior to each experiment. Sensor nodes as well as the sink record the information of each packet received and log them into the on-board flash memory. The sink broadcasts a control message at the end of each experiment. Upon receiving the message, sensor nodes start to send the logging information saved in their local memory until all logs are transmitted to the sink.

4.3 Experimental Methodology

A series of experiments is performed in this chapter to evaluate the performance of

the new protocol. In this section, the metrics used to evaluate performance, the protocols used for comparison, the experimental parameters and the experimental design are described. Section 4.3.1 introduces the evaluation metrics used in the result analysis. The four protocols used in the performance comparison study are described in Section 4.3.2. Section 4.3.3 presents the experimental parameters including the system parameters as well as the protocol parameters. Section 4.3.4 provides details on the experimental design.

4.3.1 Evaluation Metrics

The goal of the proposed protocol is to improve reliability and reduce overhead as well as latency by implementing a hop-by-hop loss recovery and detection scheme. In the experiments, the following metrics are considered when analyzing the performance of the proposed protocol:

- *End-to-End Delay*: The end-to-end delay is measured as the interval between the generation of a data packet at its source and the reception of that packet at the sink. The average end-to-end delay for each source node is calculated as the average end-to-end delay of all data packets generated by that node. The end-to-end delay shows the average amount of time it takes for the network to deliver a data packet from a particular source node to the sink.
- *Link Delay*: The link delay measures the interval from when a packet is received/created at the sender to the time it is received at the next hop receiver. Comparing the link delays is useful for understanding the network congestion level at each link as well as the impact of traffic load on a packet's link delay.
- *End-to-End Reliability*: The end-to-end reliability for each source node is defined as the number of data packets from the node that are received at the sink divided by the total number of data packets the node generates. The end-to-end reliability reflects the reliability of a given path in the network.
- *Link-level Reliability*: The link-level reliability measures the reliability of the link

between two adjacent nodes. It is defined as the number of unique data packets received by the receiver divided by the number of data packets enqueued to be transmitted by the sender at every hop.

- *Resend Rate*: As the name indicates, the resend rate is a measure of the frequency of retransmissions by a node. The resend rate for each sensor node is calculated as the number of resent data packets divided by the total number of data packets sent by the node. A higher resend rate indicates that more of the senders' transmissions at the link are unsuccessful. Since retransmitting packets may cause higher waiting time in the transmission queue, the resend rate has significant impact on both the end-to-end delay and the link delay.
- *Total Throughput:* The total throughput is measured as the number of unique data packets received at the sink divided by the time interval between when the first data packet is generated and the last packet is received. The achievable total throughput reflects the efficiency of the protocol. The higher the achievable total throughput, the faster source nodes can deliver their data packets to the sink. Both the end-to-end reliability and the end-to-end delay can affect the total throughput.
- *Link Throughput:* The link throughput measures the throughput between two neighbor nodes. Link throughput is calculated as the number of unique data packets received at the receiver divided by the time interval between when the first data packet is generated by the sender and the last packet is received by the receiver.
- *Feedback Overhead:* The feedback overhead on a link is defined as the number of feedback packets (including ACK and NACK) sent by the receiver divided by the total number of unique data packets received by the receiver (Note, however, that feedback packets are smaller than data packets).

4.3.2 Experimental Parameters

In this section, some important parameters used in the experimental study are

identified. There are two types of parameters in the experiments: system workload parameters and protocol parameters. They are introduced respectively in the following paragraphs.

System workload parameters are general system settings that affect all sensor nodes in the network. All of these parameters are adjustable according to the needs of each individual experiment.

- *Network Size* is defined as the number of source nodes participating in the experiment. It ranges from 4 to 8 in the experiments.
- *Sampling Interval* is defined as the time interval between two sensor readings. The smaller the sampling interval, the faster the sensor nodes take sensor readings and sending them. Changing the sampling interval directly affects the sending rate and the overall network throughput.
- *Number of Data Packets per Source Node* is defined as the number of sensor readings (and therefore data packets) each source node generates during the experiment. When the sampling interval is constant, the larger the number of data packet per source node, the longer the duration of the experiment.
- *Level of Interference* is defined as the extent of interference from other wireless devices using the 2.4 GHz frequency band. Two types of interference are identified in the experiments. When sensor nodes are deployed sufficiently close to each other, the radio transmissions could interfere. Another type of interference is the external interference, which is created using a separate sensor network deployed across the existing network.

There are two protocol parameters in the experiments: buffer size and sending gap. Both of them have significant impact on overall performance.

- *Buffer Size* is defined as the storage capacity in the sensor node's transmission queue, measured as the number of data packets that can be stored. The mechanism and

dequeue policy of the transmission queue are described in Section 3.3. The default buffer size of the transmission queue is 15.

- *Sending Gap* is defined as the mandatory time interval between two consecutive sending operations at the sensor node. With the default underlying MAC and TinyOS operating system running on the MicaZ testbed, if the time interval between two sending operations is below a certain threshold, the network could experience significant and unexpected packet loss [18]. Thus, In order to eliminate any potential impact of the above effect, a default 50 ms sending gap is implemented in all sensor nodes. However, the sending gap is varied in some experiments.

4.3.3 Compared Protocols

Besides the proposed data delivery protocol, four other protocols with different loss recovery and detection schemes are also implemented for the purpose of comparison.

Basic Protocol: The basic protocol has no mechanism to recover from packet loss. Sensor nodes in the basic protocol simply forward all data packets they create or receive. Since no effort is made to resend missing packets, in the basic protocol, there is no overhead for a reliability mechanism. The basic protocol is implemented as follows: when a sensor node receives a data packet, if the queue is not full, it stores the new packet in the queue. Packets are transmitted in a FIFO ordering. A transmitted packet is removed from the queue and the second packet in the queue becomes the new head of the queue if such a packet exists. After waiting the time period specified by the sending gap, the node sends the new head of the queue and repeats the above sending cycle. If the queue is full, the node discards any new received packet until a buffer space is available. The performance of the basic protocol can be used as a benchmark when studying and comparing other reliable protocols.

ACK Protocol: This protocol uses a timer-based ACK recovery scheme to detect and retransmit missing packets. A stop-and-wait explicit ACK strategy is used. After

transmitting a data packet, the sender initiates an ACK timer and waits for the feedback from the receiver. If no feedback is received before the timer expires, the sender resends the data packet. Upon receiving a data packet, the receiver generates an ACK packet and sends it back to the sender immediately. The ACK-based loss detection and recovery protocol is more aggressive in detecting missing packets compared with a NACK-based protocol, since the receiver in the ACK-based protocol confirms reception of every data packet it is received. As a result, the ACK function is widely used in the TCP/IP network as the reliable transmission scheme. However, in a wireless sensor network, where sensor nodes are extremely limited by resource and power, the heavy feedback overhead and delay generated by an ACK-based reliable transmission scheme may not be desirable.

Timer-based NACK Protocol: In most NACK loss recovery schemes, including the proposed new protocol, once the NACK packet is sent, the receiver can only passively wait for the retransmission from the sender. If, however, the NACK packet itself is lost, or the resent packet is lost again, which may be likely to occur in a congested network, the receiver can only initiate the next resend request after it receives another data packet (another out-of-order packet) from the sender. In order to reduce the waiting time and resend delay caused by the loss of a NACK packet or the retransmission, some protocols implement a NACK timer to control the sending frequency of the NACK packet. In PSFQ [32] and RMST [29] a NACK timer is implemented at the receiver. In the timer-based NACK protocols, when there is a missing data packet, the receiver aggressively sends out NACK packets to the sender for retransmission of the missing packet. If the receiver doesn't hear any reply for the retransmission request within a certain period of time, it continually resends the NACK until the data packet is recovered. In the timer-based NACK protocol implemented in this work, a sensor node starts a NACK timer immediately after it sends out a NACK packet. The node will stop the NACK timer and suppress the NACK sending process if the missing data packet is recovered before the timer is fired. When the sender receives the NACK packet, it resends all enqueued

packets for the same receivers that are newer than the last in-order received packet (as indicated in the NACK).

New Protocol with Out-of-order Buffering： As described in Section 3.6.4, a variant of the new protocol with out-of-order buffering is proposed. In the modified protocol, each node in the network maintains an out-of-order buffer besides the regular transmission queue. When receiving a data packet with a packet ID that doesn't match the local variable R[S], the receiver temporarily stores the packet in its out-of-order buffer instead of simply dropping it. The receiver checks the out-of-order buffer whenever it receives a new in-order data packet. If the previously received out-of-order packet happens to be the next expected data packet, the receiver can recover this packet. Whenever the sender receives a NACK, at least one out-of-order data packet must have been received by the receiver (otherwise the receiver won't send back the NACK). The sender therefore resends only the oldest packet among those packets following the last in-order packet received by the receivers that are still present in the senders' transmission queue. In contrast to the original proposed protocol, by introducing the out-of-order buffer, the modified protocol may be able to reduce the number of retransmissions and improve the network delay.

4.3.4 Experimental Design

In order to demonstrate the impact of different parameters to the performance of the proposed protocol, all of the experiments in this chapter are conducted by varying one parameter and keeps all other parameters unchanged. The default experimental settings are as followed. The protocol runs on a network configured as a line topology with six sensor nodes and one sink node. Each of the sensor nodes is programmed to create data packets and send them as well as packets received from its upstream neighbor, to its downstream neighbor. Each sensor node creates a new data packet every 1000 ms and has 15 buffer spaces in its queue. The default sending gap is 50 ms and each sensor node

generates 100 packets during the experiment. A summary of the default experimental settings is given in Table 4.1.

Table 4.1 Summary of Experimental Parameters

Parameters	Default Value	Range
Number of Nodes	6 nodes	4 - 8
Buffer Size	15 packets	10 - 15
Sending Gap	50 ms	40 - 50
Sending Interval	1000 ms	200 - 1000
Number of Packets Created	100 packets	50 - 200

The experiments conducted in this chapter are divided into three groups. The first group includes four different tests designed to study the performance of the new protocol under different scenarios, which includes traffic test, sampling interval test, scalability test and interference test. In the second group of experiments, the new protocol is tested under some extreme scenarios such as sampling and transmitting at five times faster rate or working with only 1/3 of the buffer space. These sets of experiments are conducted to find out when the performance of the protocol may be compromised. Finally, in the last group of experiments, the new protocol is compared with four other protocols that implement different reliability strategies, in order to demonstrate the relative strengths and weaknesses of the new protocol. For all experiments, results are studied from the perspective of end-to-end delay, link delay, total throughput, link throughput, resend rate, feedback overhead rate, link reliability and end-to-end reliability.

4.4 Basic Tests of Protocol Performance

In this section, a set of experiments is conducted to illustrate the important features of the new protocol. Since wireless sensor nodes are constrained by both bandwidth and storage space, the buffer size and the sampling interval play very important roles in the performance of any reliable protocol. Other system parameters such as network traffic,

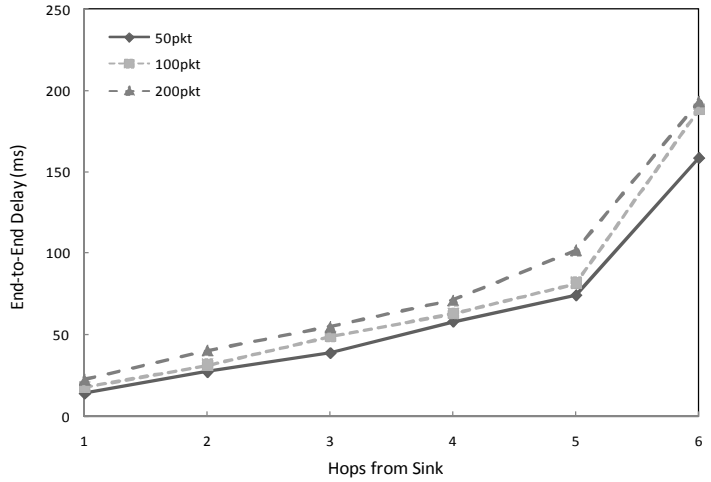
interference between sensor nodes and network size may also have impact on the performance of the protocol. Four separate tests are performed in this section to discuss how system and protocol parameters may affect the performance of the protocol.

4.4.1 Traffic Test

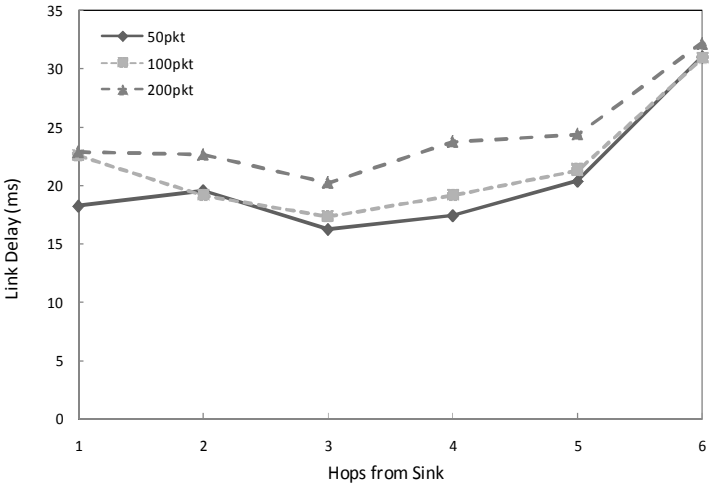
In the first set of experiments, the performance of the new protocol with different number of packets per source node is tested. Since each sensor node creates new packet with a fixed sampling interval, the total number of packets created by each node is determined by the duration of the experiment. The traffic test illustrates basic performance properties of the protocol. The default experimental settings are used in the traffic test, except the number of packets created per node varies in individual experiments. The experiment starts with generating 50 packets per node, which equals to 300 packets in total in the network. Then the number of packets created (and the experiment duration) is increased by 100% to 100 packets per node, which equals to 600 packets in total. At last, the number of packets created is increased by 300% to 200 packets per node, which equals to 1200 packets in total. The experiment results are shown in Figure 4.1, Figure 4.2 and Table 4.2.

The first observation of the result from Table 4.2 is that reliability is 100% (the figure of link level reliability is not presented here because the reliability at each link is 100%), which indicate all data packets generated by the sensor nodes are successfully transmitted to the sink. This result is encouraging because the protocol meets the most important design objective: 100% reliable data delivery.

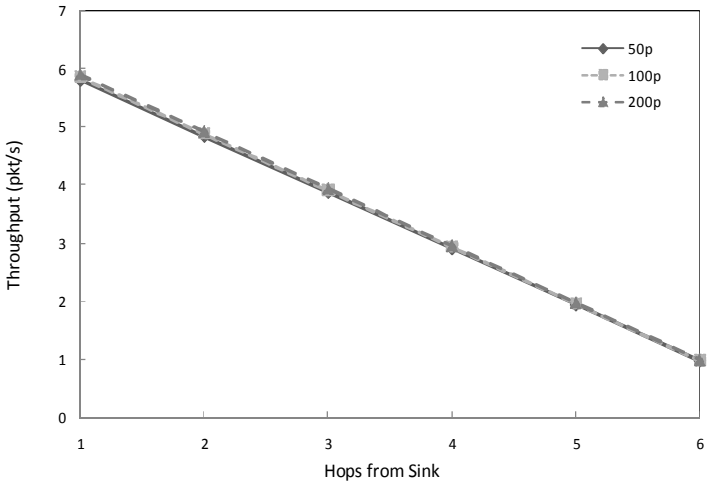
Figure 4.2(a) represents the resend rate per hop during the traffic test. A observation can be made that on average only around 5.3% of the data packets incurred hop-by-hop retransmission, and highest overhead is still less than 8.3%. The feedback overhead presented in Figure 4.2(b) contains both NACK and ACK packets. All three experiments show similar feedback overhead rate at each hop.



(a) End-to-end Delay

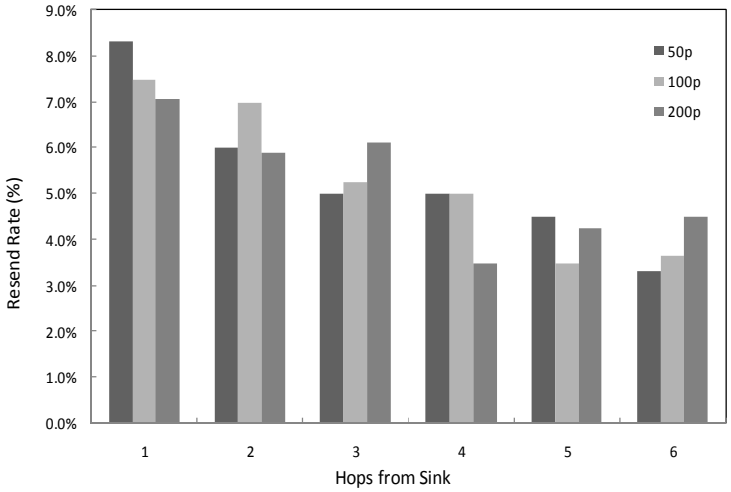


(b) Link Delay

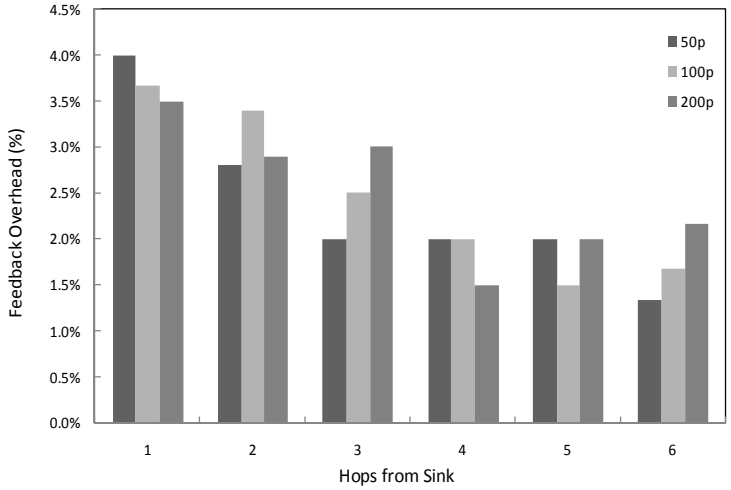


(c) Throughput

Figure 4.1 Throughput and Delay in Traffic Test



(a) Resend Rate



(b) Feedback Overhead

Figure 4.2 Overhead Costs in Traffic Test

Table 4.2 Results of Traffic Test

Experiment	End-to-End Reliability	Total Throughput (pkt/sec)	Average Resend Rate	Average Feedback Overhead
50pkt	100%	5.80	5.4%	2.4%
100pkt	100%	5.87	5.3%	2.5%
200pkt	100%	5.9	5.2%	2.5%

When working with different traffic load, the network is only transmitting packets for a different length of time, while the number of packet transmitted per second at each node remains the same. As can be seen from Figure 4.1 (c), the throughput at each hop

for all three experiments are almost identical and the total throughput of the network as shown in Table 4.1 are at the same level as well.

From Figure 4.1 (a), an increasing tread of the end-to-end delay can be observed as the number of hops from the sink increase. For example, the end-to-end delay of packets created by node 6 is much larger than the end-to-end delay of packets created by nodes. The above observation is because data packets generated by node 6 need to travel six hops before they can reach the sink, and thus they will take much longer time compare with packet from node 1, which is only one hop away from the sink. Another observation that can be made from Figure 4.1 (a) is that the relationship between the hops from the sink and the end-to-end delay is non-linear. For example, in the 200 packet experiment, the delay of packets from node 1 is 22.38 ms; the delay of packets from node 2 is 40.41 ms, which is slightly less than twice of node 1's packet delay; the delay of packets from node 6 is 193.13 ms, which is nine times larger than node 1's packet delay. The above observation is likely because packets from the nodes far from the sink usually end up at the ends of the queues of the intermediate nodes, which increase the queueing delay. And if there are no following packets of these packets, it will take long time to for receivers to detect loss. Another possible reason for the larger delay of packets from node 6 is the larger possibility of link interference of these packets compared with packets from nodes closer to the sink.

Figure 4.1(b) and Figure 4.2(a) plot the link delay and link resend rate at each hop in the network. Often, on a given link, the higher the resend rate, the larger the link delay. Recall that the new protocol is designed upon a NACK-based loss detection and recovery mechanism. As described in Section 3.4, in the new protocol, a packet loss can only be detected when an out-of-order packet is received by the receiver. Thus, if a packet is lost during the transmission, the period of time it waits in the queue for retransmission will certainly introduce significant amount of delay to the node's average link delay. The positive correlation between resend rate and link delay can be observed from Figure 4.1.

For example, in Figure 4.2 (a), the resend rate at node 3 increases as more packets are generated. Accordingly, from Figure 4.1 (b), the link delay also increases at node 3 if compares the 50 packet line, the 100 packet line and the 200 packet line.

However, the resend rate and the link delay on a given link are not always directly related. Comparing node 4's link delay and resend rate from Figure 4.1 (b) and Figure 4.2 (a), although the 50 packet experiment shares the same resend rate with 100 packet experiment, it shows lower link delay than 100 packet experiment does. The reason behind this observation is that, the pattern of retransmission may also influence the positive relationship between the resend rate and the link delay. For example, consider the case where two packets were missing and need retransmissions at node 6. It is possible that these two packets lost are independent events. Assuming both of them are recovered successfully at the first resend attempt, the total extra delay caused by the retransmission is two sampling intervals, as explained in the previous paragraphs. However, it is also possible that these two packets are adjacent packets and both of them get lost during their initial retransmissions. This is a possible scenario that may be caused by network congestions or packet collisions. In this case, the first lost packet won't be scheduled for retransmission until the third data packet gets received by the receivers. The first packet has to wait at least two sampling intervals in the transmission queue, while the second lost packet needs to wait one sampling interval. Thus, with same resend rate, in the second scenario, the total extra delay caused by the retransmission is three sampling interval. It is reasonable to believe that under the same per node throughput rate, the longer the experiment, the higher the possibility to have consecutive packet loss in the experiment, which explains why the 50 packet experiment shows lower link delay than 100 packet experiment at node 4.

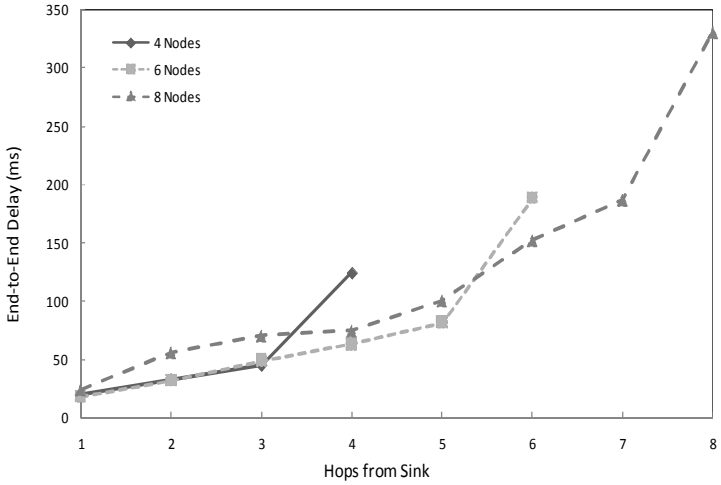
For a single experiment, the link delay is not only influenced by the resend rate, but also depends on the link's distance from the sink. For example, in the 200 packet experiment, although node 6 has much lower resend rate than node 1, it can be observed

from Figure 4.1 (b) that node 6's link delay is still higher than node 1. As explained in the previous paragraph, both packet loss and retransmission have significant impact on the link delay. However, the level of impact varies at different links. At node 6, the node only sends one packet per 1000 ms to its downstream neighbor node 5. When a packet from node 6 gets lost, node 5 won't be able to detect the gap in the incoming packet until the next packet arrives. Thus, the lost packet has to wait in the transmission queue for at least one sampling interval (1000 ms in the traffic test) to get retransmitted. However, if a packet loss occurs at node 1, it will be different. Because node 1 not only sends its own generated packets but also forwarding all received packets from previous nodes, it actually sends at a speed of six packets per second to the sink, a much faster rate than node 6. The lost packet from node 1 only needs to wait approximately one sixth of a sampling interval to get resent. Therefore, as shown in Figure 4.1, the higher resend rate doesn't necessarily lead to higher link delay if comparing resend rate at different nodes.

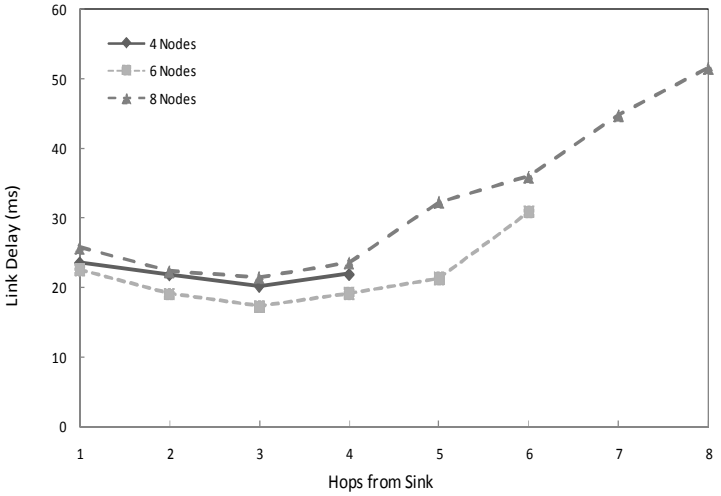
In summary, the protocol can provide 100% reliable data delivery while maintain a relatively low network overhead. Increasing the number of packets generated per node only extends the length of the experiment which has little impact on the protocol's overall performance. At the same time, because of the higher possibility of adjacent packet loss, a slightly larger end-to-end delay and link delay can be observed when more packets are generated per node.

4.4.2 Scalability Test

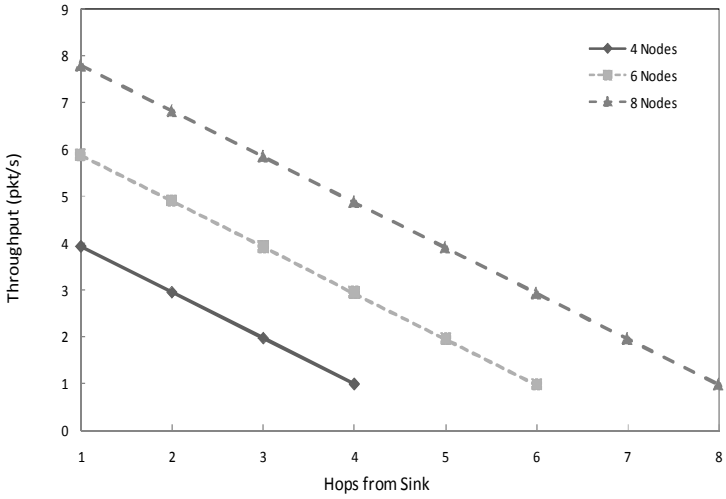
The traffic test demonstrates the performance of the new protocol with different experimental durations. The next question to address is: how does the performance of the new protocol scale with the number of nodes in the network path to the sink? In this section, the proposed protocol is tested with a path length of four nodes, six nodes and eight nodes respectively. The default value is used for other experimental parameters. The test results are shown in Figure 4.3, Figure 4.4 and Table 4.3.



(a) End-to-end Delay

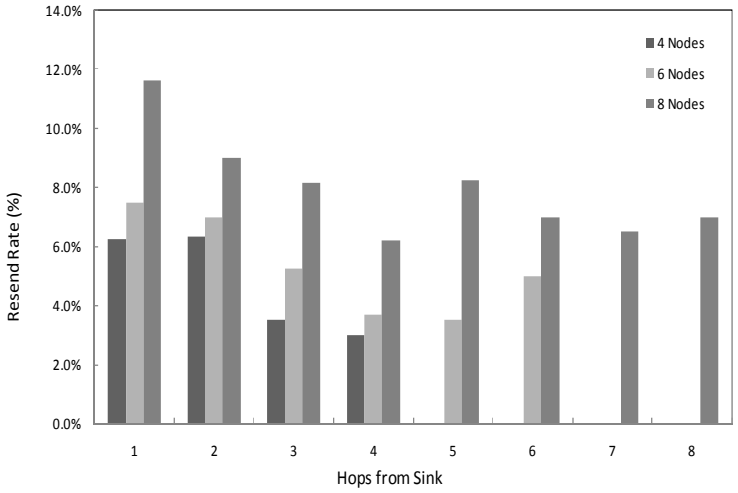


(b) Link Delay

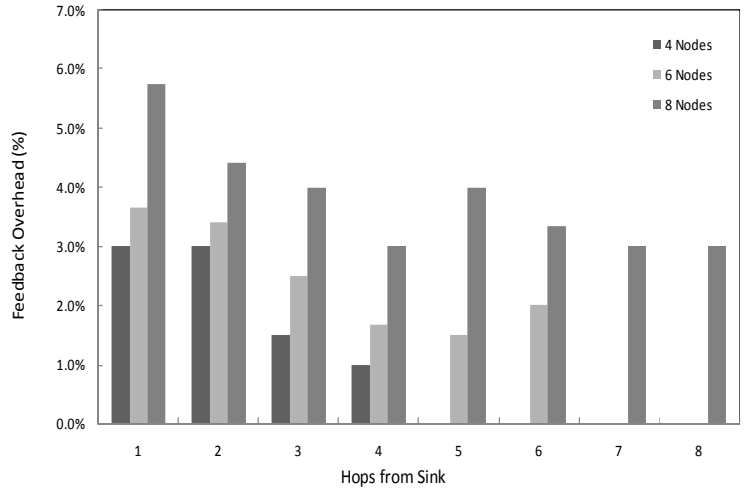


(c) Throughput

Figure 4.3 Throughput and Delay in Scalability Test



(a) Resend Rate



(b) Feedback Overhead

Figure 4.4 Overhead Costs in Scalability Test

Table 4.3 Results of Scalability Test

Experiment	End-to-End Reliability	Total Throughput (pkt/sec)	Average Resend Rate	Average Feedback Overhead
4 Nodes	100%	3.93	4.8%	2.1%
6 Nodes	100%	5.87	5.3%	2.5%
8 Nodes	100%	7.79	8.0%	3.8%

The end-to-end reliability and the link reliability are both 100% in all three experiments. As the network size increases, there are more nodes creating data packets. In the four nodes experiment, a total number of 1069 packets are sent by all nodes, 1048 of

which are data packets (including retransmission packets) and 21 are feedback packets. In the six node experiment, there are 2263 packets sent in total, including 2211 data packets and 52 feedback packets. The number of packets sent increases to 4024 in the eight node experiment, 3888 of them are data packets and 136 of them are feedback packets.

As the network size increases, the network throughput increases as well. A linear relationship between the network size and throughput is shown in Figure 4.3(c). The throughput at node 1 is around four packets per second in the four node experiment. The throughput doubled to approximately eight packets per second at node 1 when the network size increased by 100% to eight nodes. Since the total transmission time remains unchanged under different network sizes, the linear relationship observed in Figure 4.3(c) is a result of the increasing number of data packets generated in the network. The total throughput of the experiments is presented in Table 4.3. The total throughput rises by 49% when the network size increases from four to six, and rises by another 33% when the network size increases from six to eight. The results are in line with expectations.

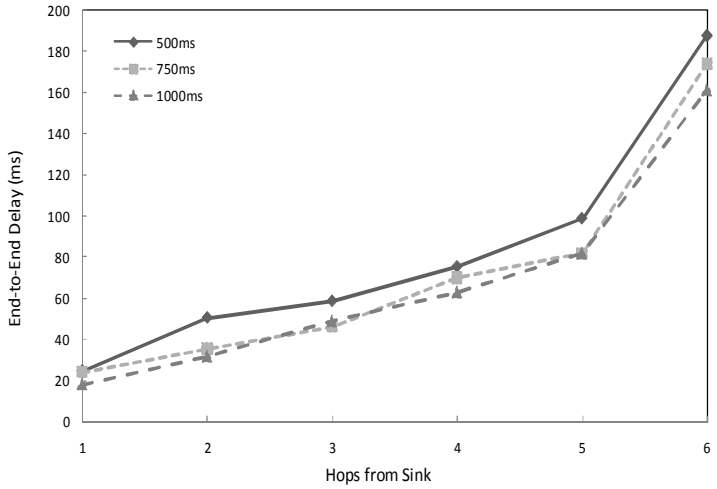
The resend rate and overhead also exhibit an increasing trend as the network size increase as shown in Figure 4.4(a) and Figure 4.4(b). The total resend rate in the six node experiment is 12% higher than in the four node experiment and the total resend rate in the eight nodes experiment is 68% higher than in the four nodes experiment. This result is consistent with expectations. When more data packets are transmitted in the network within the same time period, sensor nodes have a higher possibility to experience packet collisions and network congestion. Thus the positive relationship between network size and resend rate is reasonable.

In summary, the three experiments in the scalability test show that the new protocol can provide 100% reliable data delivery while maintaining a relatively low network overhead for various path lengths. Increasing the number of sensor nodes in the path to the sink will result in higher throughput, higher resend rate and higher overhead. The impact of path length on end-to-end delay and link delay can also be observed.

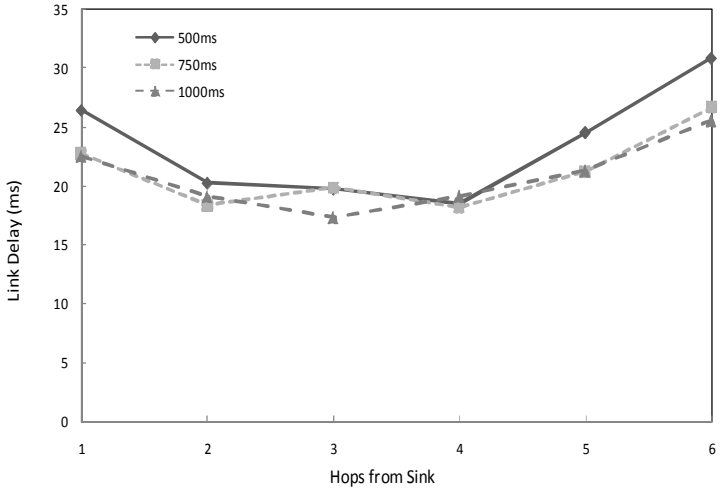
4.4.3 Sampling Interval Test

The next set of experiments studies the impact of the sampling interval on the protocol's performance. The sampling interval is the time interval between two sensor readings at each source node. With a smaller sampling interval, sensor nodes create more data packets per time unit, possibly creating more network congestion since the transmission time required for each packet remains the same. The default experimental settings are used in the sampling interval test, except the sampling interval varies in different experiments. In the first experiment, the sampling interval is one sample every 1000 ms. The sampling interval is decreased by 25% to one sample every 750 ms in the second experiment. At last, the sampling interval is further decreased by 50% to one sample every 500 ms in the third experiment. The test results are shown in Figure 4.5, Figure 4.6 and Table 4.4.

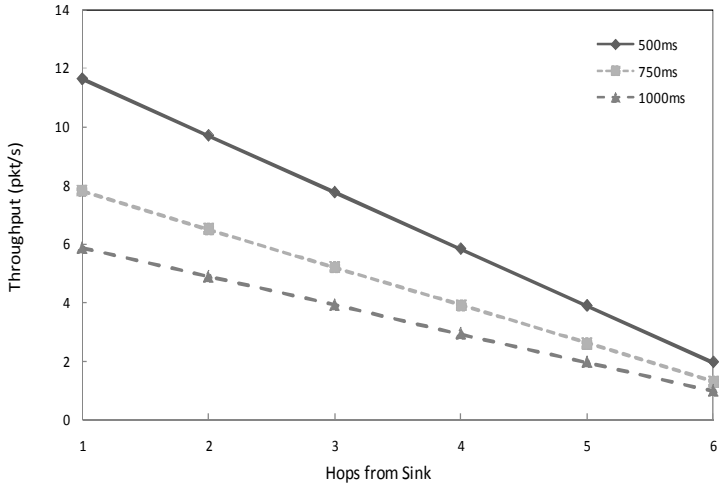
As in the previous test, no packet loss is observed in all three experiments at both the end-to-end level and the link level. However, note in Figure 4.6(a) that there is an increasing trend in the resend rate as the sampling interval becomes smaller. The 500 ms experiment has a resend rate approximately 54% higher than in the 750 ms experiment, and 92% higher than in the 1000 ms experiment. With a smaller sampling interval, sensor nodes are simply creating more data packets per second. For example, in the 1000 ms experiment, each node generates one data packet per second. Sensor nodes also need to forward packets they receive from their upstream neighbors. Thus, node 1 needs to handle at least six data packets per second. In the 500 ms experiment, since each node creates two data packets per second, the sending rate at node 1 is at least twelve packets per second. The higher resend rate observed in the 500 ms experiment is likely due to the higher loss rate, which is a result of smaller sampling interval. It is encouraging to observe that the proposed protocol is able to maintain high reliability in the higher-rate environment. However, it is reasonable to predict that if the sampling interval is further



(a) End-to-end Delay

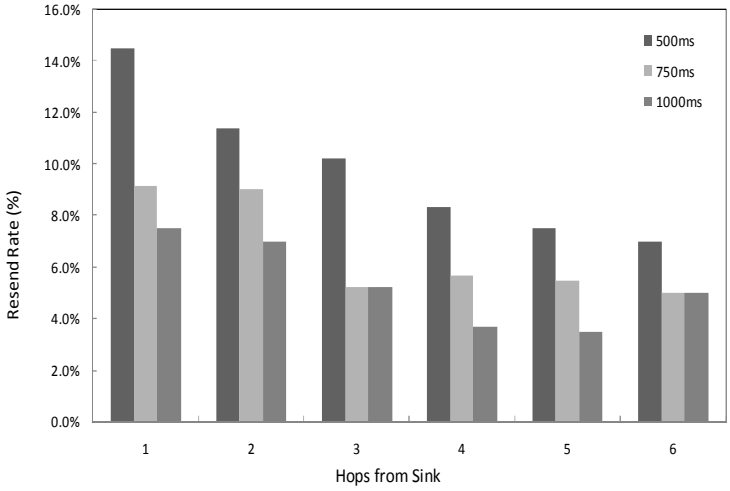


(b) Link Delay

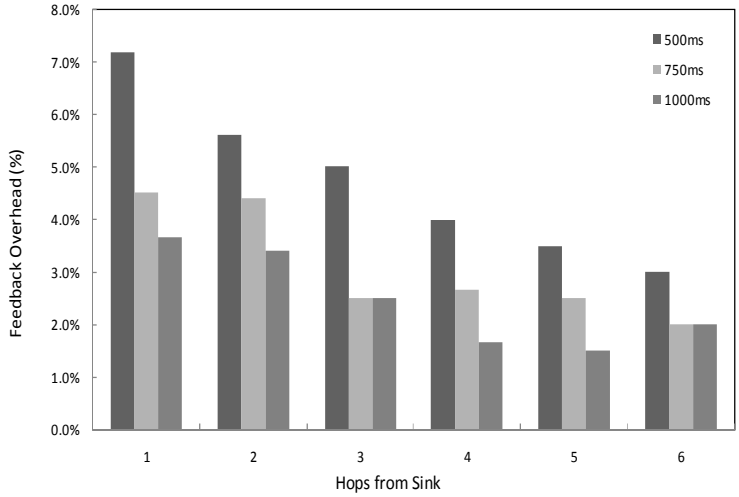


(c) Throughput

Figure 4.5 Throughput and Delay in Sampling Interval Test



(a) Resend Rate



(b) Feedback Overhead

Figure 4.6 Overhead Costs in Sampling Interval Test

Table 4.4 Results of Sampling Interval Test

Experiment	End-to-End Reliability	Total Throughput (pkt/sec)	Average Resend Rate	Average Feedback Overhead
500 ms	100%	11.63	9.8%	4.7%
750 ms	100%	7.78	6.6%	3.1%
1000 ms	100%	5.87	5.3%	2.5%

decreased to a level where a sensor node is receiving more packets than it can handle, packet loss can be inevitable, considering the limited storage space in sensor nodes. A detailed discussion of the impact of a low sampling interval on the end-to-end reliability

is presented in Section 4.5.1.

Figure 4.5(c) presents the throughput per node in all three experiments. As the sampling interval becomes smaller, the sensor nodes are sending more data packets per second and thus an increasing trend of throughput is observed. The result in Table 4.4 shows that the total network throughput in the 500 ms experiment is nearly 49.5% higher than in the 750 ms experiment and is 99% higher than in the 1000 ms experiment.

The end-to-end delay and link delay are plotted in Figure 4.5(a) and Figure 4.5(b), respectively. The 750 ms experiment shows slightly higher end-to-end delay and link delay than the 1000 ms experiment. The sensor nodes in the 500 ms experiment are sampling two times faster than in the 1000 ms experiment and 50% faster than in the 750 ms experiment, and the end-to-end delay is higher than in the other two. All else equal, it is reasonable to assume that the faster nodes send, the smaller the delay. However, as shown in this test, the smaller sampling interval leads to higher resend rate, which in general has negative impact on the delay, as discussed in Section 4.4.1. Thus there is a tradeoff between the impact of sampling interval and the resend rate to the delay. As shown in the experimental results, at the sampling interval range between one packet per second to two packet per second, the resend rate plays a more important role in the end-to-end delay.

In summary, according to the three experiments in the sampling interval test, it can be concluded that the protocol can achieve desirable reliability and low overhead under different sampling intervals. Decreasing the sampling interval within the range considered here has no significant impact on the end-to-end reliability and link reliability. At the same time, the network throughput is increasing because more data packets are pumped into the network. The resend rate and overhead are also increasing due to the higher possibility of packet loss.

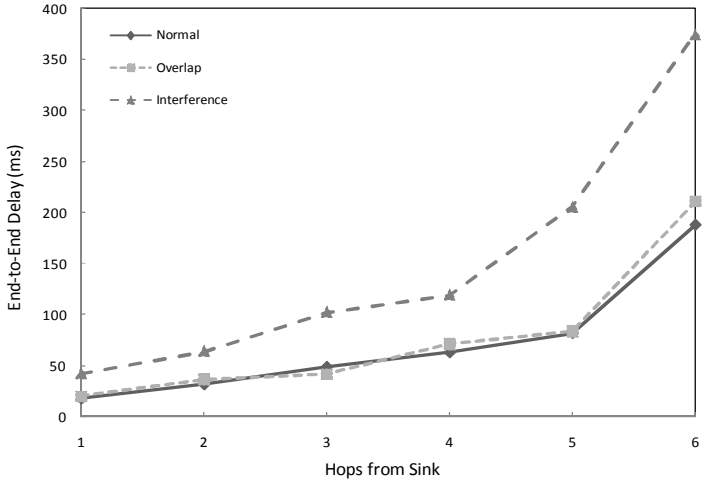
4.4.4 Interference Test

In WSN applications, sensor nodes may be required to work in various environments in which interference between sensor nodes is unavoidable. In the last test, the impact of interference on the protocol's performance is tested. The default experimental settings described in Section 4.3.4 are used in this test. In the first experiment ("Normal"), sensor nodes are deployed in a linear topology and each node is one hop away from its neighbor nodes as in the previous tests. In the second experiment ("Overlap"), nodes were moved close to each other that every node can cover four nodes in its radio range. In the final experiment ("Interference"), four other nodes were added into the network. By defining different group ID (a bit in the packet header), those other nodes form two pairs and can only communicate with each other. These nodes will not join in the existing network. Each of the two nodes is deployed on one side of the original network. The other nodes keep transmitting during the entire experiment to each other at the rate of 20 packets per second. The experimental results are shown in Figure 4.7, Figure 4.8 and Table 4.5.

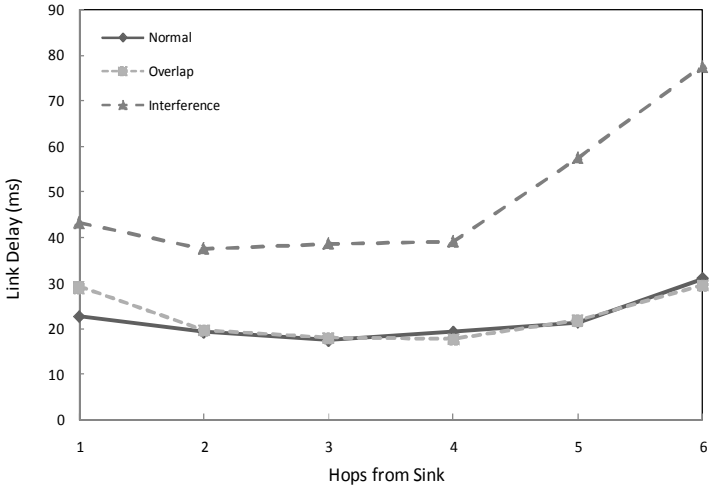
From Figure 4.7(a), significant differences in the end-to-end delay between the interference experiment and other two experiments can be observed. The frequent data transmission between the four new nodes appears to have strongly interfered with the normal communication among the other nodes. The overlap experiment also exhibits slightly higher end-to-end delay than the normal experiment. It is likely that the closer distance between sensor nodes in the overlap experiment causes interference and contention to occur and thus increases the end-to-end delay.

Figure 4.8(a) and Figure 4.8(b) plot the resend rate and overhead respectively. The average resend rate for the three experiments are 5.3%, 6.1% and 13%, respectively. The interference experiment shows 145% higher resend rate than the normal experiment and 109% higher than the overlap experiment.

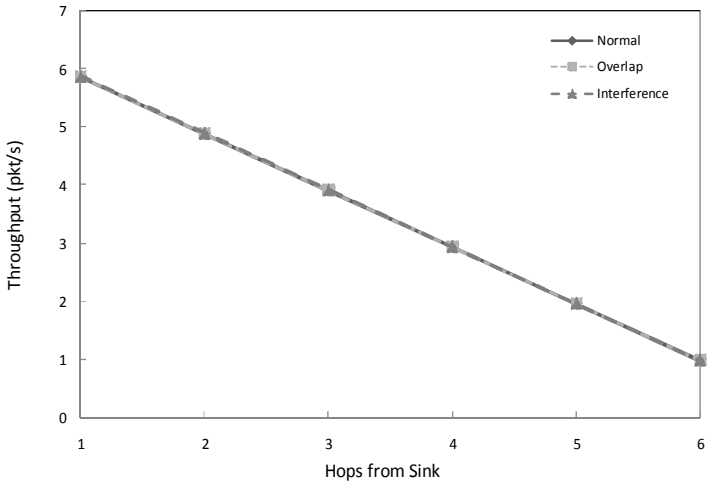
Because all experiments run with a 1000 ms sampling interval experience no packet



(a) End-to-end Delay

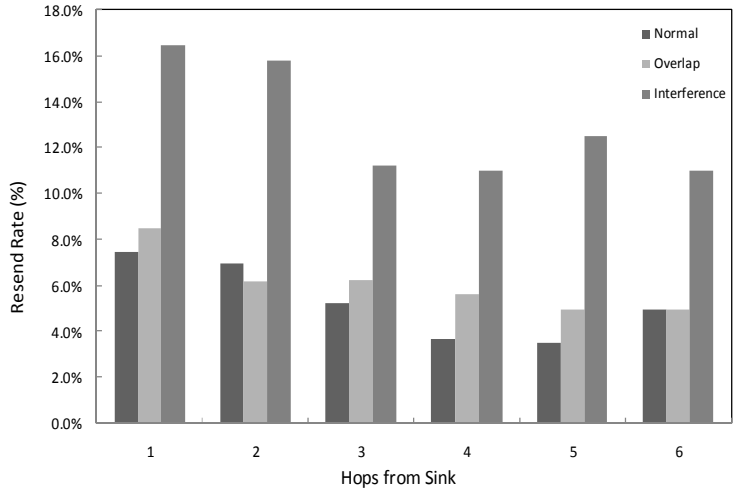


(b) Link Delay

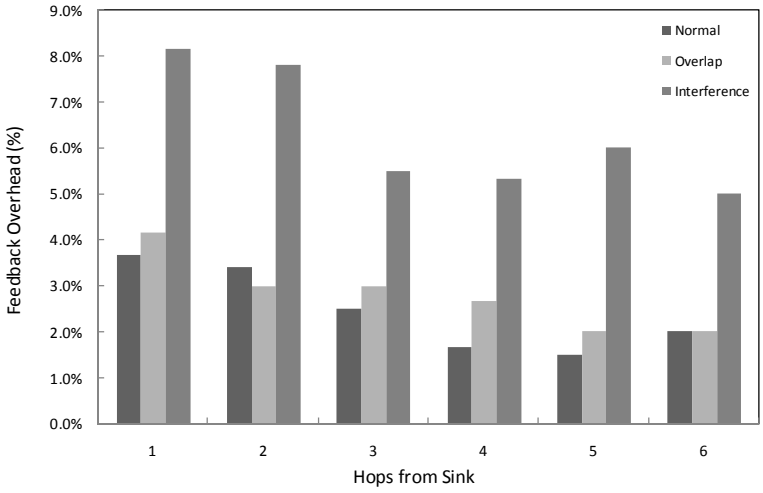


(c) Throughput

Figure 4.7 Throughput and Delay in Interference Test



(a) Resend Rate



(b) Feedback Overhead

Figure 4.8 Overhead Costs in Interference Test

Table 4.5 Results of Interference Test

Experiment	End-to-End Reliability	Total Throughput (pkt/sec)	Average Resend Rate	Average Feedback Overhead
Normal	100%	5.86	5.3%	2.5%
Overlap	100%	5.87	6.1%	2.8%
Interference	100%	5.87	13%	6.3%

loss, it is reasonable to predict that all experiments should have approximately the same throughput. The throughput results shown in Figure 4.7(c) and Table 4.5 confirm this prediction. There is only 0.05% variance in the throughput among the three experiments.

In summary, interference from other traffic flows can significantly disrupt communication. The interference may lead to higher end-to-end delay, higher link delay and a higher resend rate. Interference among the nodes on the same path to the sink also exists, although the impact is not as strong as from other traffic flows. Interference has no significant impact on the throughput of the network.

4.5 Protocol Stress Tests

The performance of the new protocol has already been tested with various system and protocol parameters in Section 4.4. The protocol achieves 100% reliability in all of the tests. However, an increasing trend of resend rate is noticed with the increase of system and protocol parameters. In this section, the performance of the protocol is studied under some extreme conditions where the performance may be compromised. The reminder of this section is organized as follow: Section 4.5.1 studies the impact of the sending gap and the sampling interval on the new protocol. Section 4.5.2 evaluates the performance of the protocol with different buffer sizes.

4.5.1 Effect of Sending Gap and Sampling Interval

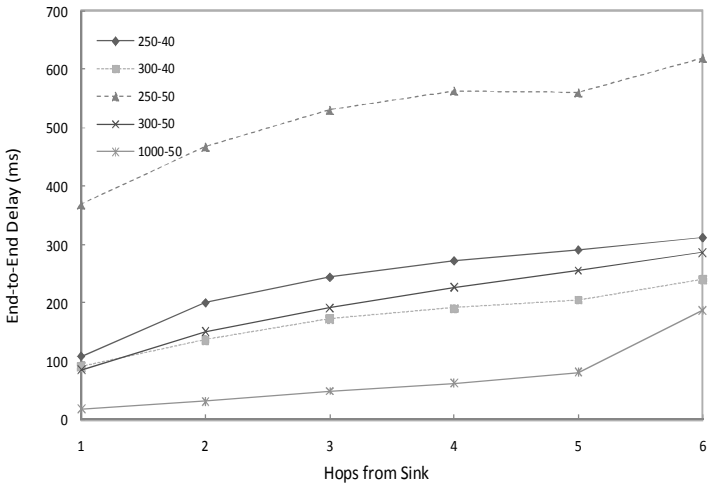
One significant difference between simulation and testbed implementation is that simulation simplifies some assumptions that cannot be ignored in the implementation. As tested in Flush [24], with the default underlying MAC and TinyOS operating system running on the MicaZ testbed, if the packet interval between two sending operations is below a certain threshold, the network could experience significant and unexpected packet loss. Thus, in the implementation of the protocol, in order to achieve the best possible performance and completely remove the impact of the above issue, a 50 ms sending gap between two consecutive sending operations is implemented. In other words, after sending a packet, the sensor node is forced to wait 50 ms before it can send another packet. By implementing the sending gap, the maximum sending rate a sensor node can

reach is limited. A 50 ms sending gap implies that a sensor node can send no faster than 20 packets per second. Since each sensor node in the network has limited storage space, when the receiving rate at a node is larger than the sending rate, packet loss is inevitable after the buffer gets filled up. The receiving rate in this protocol is determined by the speed at which the sensor nodes generate new data packets, which is decided by the sampling interval. Thus, in this section, several experiments are conducted to study the effect of different sending gap and sampling interval on the performance of the new protocol. The experimental settings of the four experiments conducted in this test are the default settings with different combinations of the sending gap and the sampling interval. The sending gap and the sampling interval of the experiments are summarized in Table 4.6. Figure 4.9, Figure 4.10 Figure 4.11 and Table 4.7 show the experimental results. The result of the experiment with default 1000 ms sampling interval and 50 ms sending gap is also included for comparison.

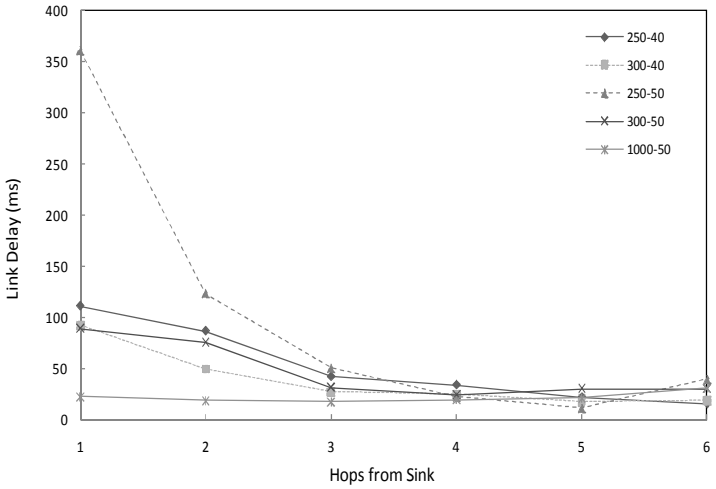
Table 4.6 Experiment Settings of Sending Gap and Sampling Interval Test

Experiment	Sending Gap	Sampling Interval
250-40	40 ms	250 ms
300-40	40 ms	300 ms
250-50	50 ms	250 ms
300-50	50 ms	300 ms

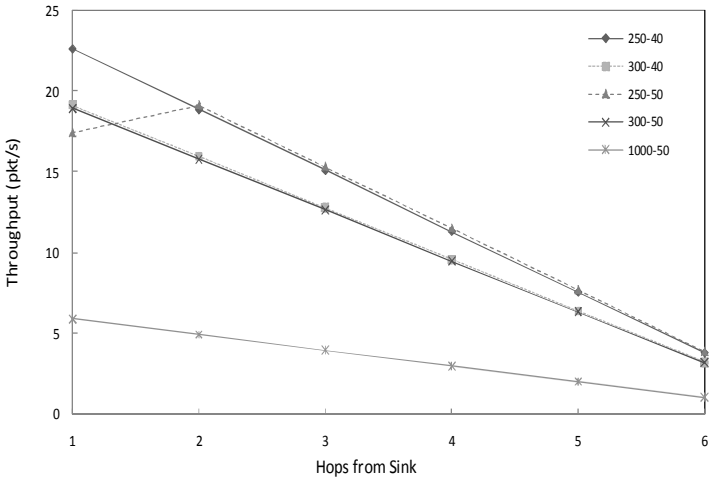
Figure 4.9(a) and Figure 4.9(b) show the end-to-end delay and link delay, respectively. Because of heavy packet loss and retransmissions at link 1, the 250 ms sampling interval and 50 ms sending gap experiment (250-50) shows significantly higher delay than all other experiments at both end-to-end level and link level. Experiments with larger sampling interval and with same sending gap have lower end-to-end delay. As shown in Figure 4.6(a), the 300-40 experiment exhibits lower end-to-end delay at every node compared with the 250-40 experiment. The rationale behind this observation is that although a lower sampling interval can reduce the amount of time a packet must wait in



(a) End-to-end Delay

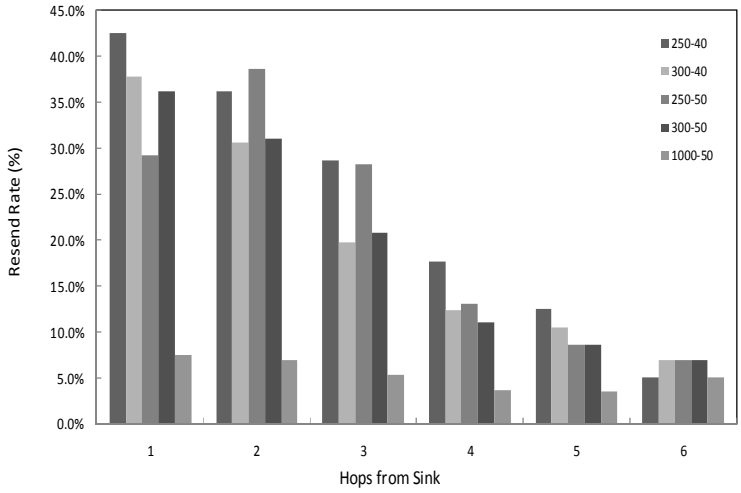


(b) Link Delay

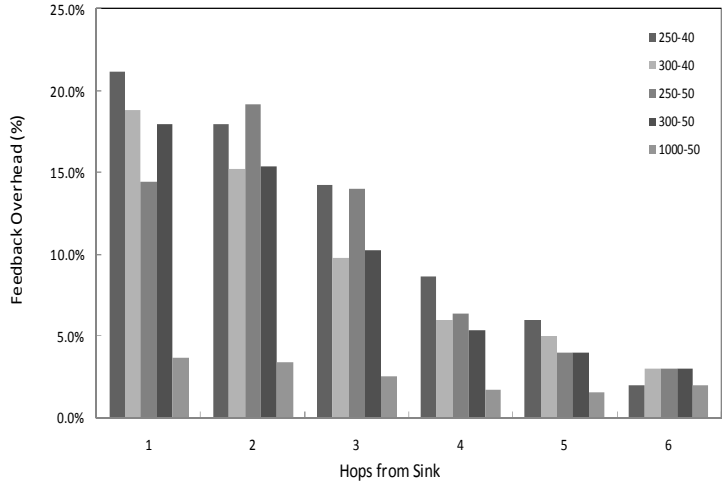


(c) Throughput

Figure 4.9 Throughput and Delay in Sending Gap and Sampling Interval Test



(a) Resend Rate

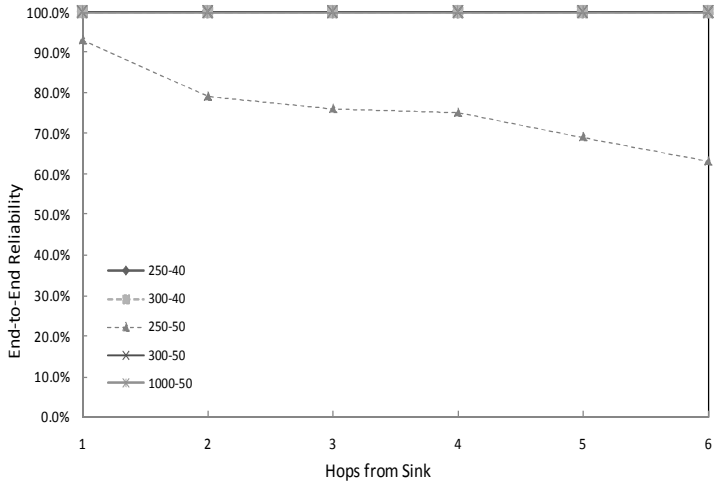


(b) Feedback Overhead

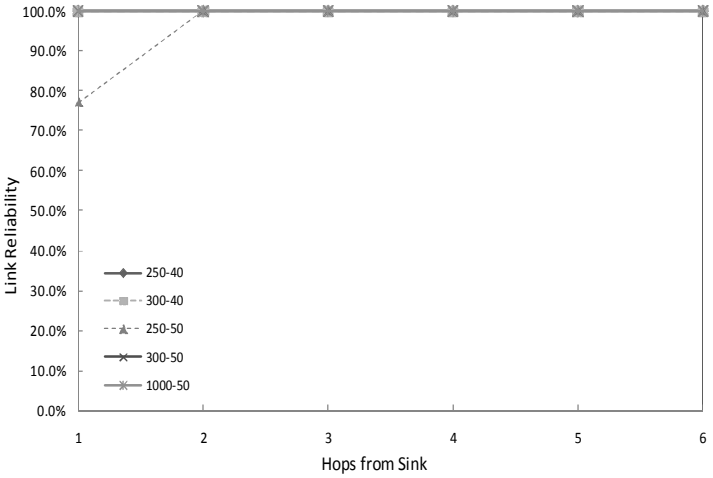
Figure 4.10 Overhead Costs in Sending Gap and Sampling Interval Test

the queue for retransmission, it also results in a higher resend rate, which in turn leads to higher end-to-end delay. It is also observed that the 300-40 experiment has smaller delay than the 300-50 experiment, which has the same sampling interval but a larger sending gap. The result is consistent with expectations. All else being equal, the faster the nodes send, the smaller the overall delay.

Figure 4.9(c) presents the per node throughput of the experiment. Without packet loss, it is observed that with the same sampling interval, the throughputs are nearly identical, even if the sending gap is different. For example, the throughputs in the 300-40



(a) End-to-end Reliability



(b) Link Reliability

Figure 4.11 Reliability in Sending Gap and Sampling Interval Test

Table 4.7 Results of Sending Gap and Sampling Interval Test

Experiment	End-to-End Reliability	Total Throughput (pkt/sec)	Average Resend Rate	Average Feedback Overhead
250-40	100%	22.61	23.8%	11.7%
300-40	100%	19.12	19.7%	9.6%
250-50	77%	16.92	20.8%	10.2%
300-50	100%	18.92	19.1%	9.3%
1000-50	100%	5.87	5.3%	2.5%

and 300-50 experiment almost overlap with each other in Figure 4.9(c). Even with packet loss, the throughput in the 250-40 and 250-50 experiments is still very close at links without packet loss, such as from link 2 to link 6. The plunge of the throughput at link 1 in the 250-50 experiment is mainly due to the large packet loss at node 1. Thanks to the small sampling interval, all experiments show much higher network throughput than the 1000 ms sampling interval experiment.

Figure 4.11(a) and Figure 4.11(b) plot the end-to-end and the link reliability of the experiments, respectively. Packet loss is observed in the experiment. As shown in Figure 4.11(b), all sensor nodes are able to maintain 100% reliability except node 1 in the 250 ms sampling interval and 50 sending rate experiment (250-50).An interesting comparison is that both 250 ms sampling interval and 40 ms sending gap experiment (250-40) and 300 ms sampling interval and 50 ms sending gap experiment (300-50) are still able to achieve 100% reliability. An explanation as to why packet loss only occurs in 250-50 experiment is as follows. The 250 ms sampling interval implies that each sensor node generates four packets per second. Since all sensor nodes except the very last one need to forward the incoming data packet from their upstream neighbor, the minimum number of data packets every sensor node needs to transmit per second can be calculated. In Table 4.8, the number of data packets that need to be sent in the 250-50 experiment, assuming no packet loss, is presented. As one can see, node 1 gets 24 data packets every second coming to the transmission queue. However, since the sending rate is only 20 pkt/s as determined by the sending gap, node 1 is receiving more packets in its queue than it can send out. When the receiving rate exceeds the sending rate, the data packet cannot be sent immediately is stored in the transmission queue. When the queue fills up, the node is then forced to drop data packets. That explains the larger number of packets lost on the link between node 1 and the sink. A similar argument also explains why the 250 ms sampling interval and 40 ms sending gap didn't have any packet loss. Since the 40 ms sending gap implies a maximum sending rate of 25 packets second, with the same sampling interval

as the 250-50 experiment, node 1 in the 250-40 experiment is sending fast enough to handle all incoming packets and thus has no packet loss.

Table 4.8 Sending Rate and Receiving Rate of Sending Gap and Sampling Interval Test

Category	Node 1	Node 2	Node 3	Node 4	Node 5	Node 6
Data Packet Generated	100	100	100	100	100	100
Data Packet Received	500	400	300	200	100	0
Data Packet in Total	600	500	400	300	100	100
Packet need to Be Sent	24 pkt/s	20 pkt/s	16 pkt/s	12 pkt/s	8 pkt/s	4 pkt/s

Figure 4.10(a) and Figure 4.10(b) present the resend rate and overhead of the experiments. Some observations of the results are summarized and discussed as follows:

- With the same sampling interval, the experiment with smaller sending gap has slightly higher resend rate. For example, the average resend rate for the 300-40 experiment is 19.7%, around 2.5% higher than the 300-50 experiment. A smaller sending gap implies a higher sending rate. In general, the faster a node is sending, the higher the possibility that the node may experience packet collisions and channel contentions, which may in turn leads to a higher packet loss rate and a higher resend rate.
- With the same sending gap, the experiment with smaller sampling interval has a higher resend rate. For example, the average resend rate in the 250-40 experiment is about 19% higher than in the 300-40 experiment. This result is in line with the findings in the sampling interval test in Section 4.4.3. With a 250 ms sampling interval, a single sensor node generates four packets per second. In a six node network, 24 packets are created and transmitted in the network every second. The number of packets for the 300 ms sampling interval can be calculated with the same method. Comparing with the 300-40 experiment, the 250-40 experiment transmits 20% more packets per second. The more packets transmitted in the network, the higher the packet loss rate and the resend rate.

- An increasing trend of the link level resend rate can be observed in Figure4.10 (a) as the distance to the sink decreases. This observation is reasonable based on the fact that nodes closer to the sink have more packets to forward and experience greater network contention. Node 1 shows the highest resend rate among all nodes in almost all the experiments in this test. However, an interesting observation about the resend rate with experiment has packet loss is that the there is a plunge in the resend rate at the link with packet loss. In the 250-50 experiment, node 1's resend rate is only 28.4%, compared with 37.1% at node 2. As discussed in previous sections, the packet loss in experiment 250-50 is because node 1 receives more packets than it can transmit. Thus when a NACK is received by node 1, it is possible that the data packet that would otherwise be retransmitted has already been removed from the queue due to the limited storage space. As described in Section 3.5, instead of resending the missing packet (which is no longer available), node 1 will set the SKIP field to 1 and transmitting all packets available in the queue to the receiver. As a result the resend rate is lower at node 1.

The effect of the sampling interval and the sending gap to the performance of the protocol is quite significant. By comparing the 250 ms sampling interval and 40 sending rate experiment (250-40) and the default 1000 ms sampling interval and 50 sending rate experiment (1000-50), one can observe that the 250-40 experiment has on average 235% higher end-to-end delay and 390% higher total resend rate than the 1000-50 experiment. On the other hand, with a lower sampling interval and a lower sending gap the network throughput can be significantly improved. The 250-40 experiment shows 305% higher total throughput than the 1000-50 experiment.

4.5.2 Effect of Buffer Size

One of the characteristics of wireless sensor networks is that sensor nodes have limited storage space. In this section, four experiments are performed to study the impact

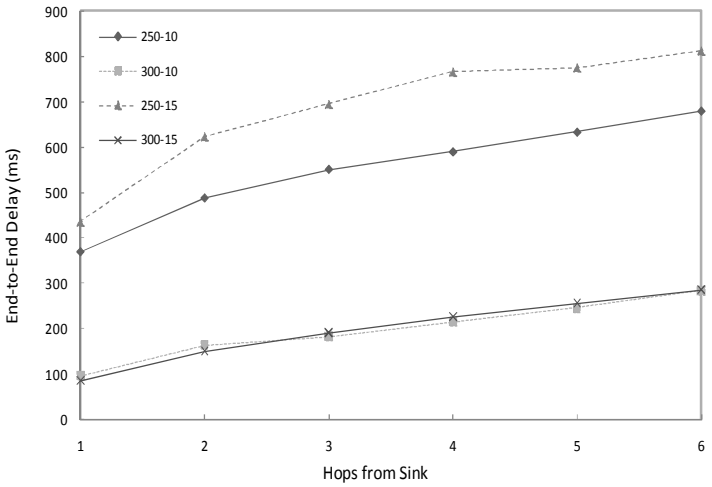
of buffer size to the performance of the proposed protocol. The default experimental settings are used in this test. In order to study the effect of buffer size in the case with and without packet loss, the experiments conducted in this test are run with different sampling interval and buffer size. The experiment results are presented in Figure 4.12, Figure 4.13, Figure 4.14 and Table 4.10.

Table 4.9 Experiment Settings of Buffer Size Test

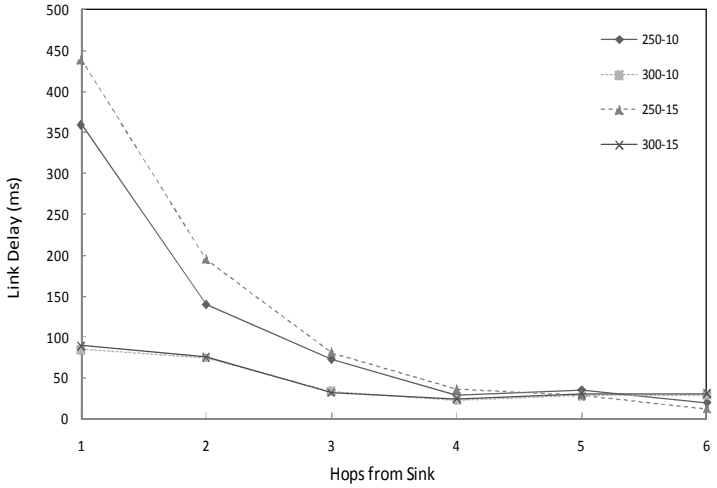
Experiment	Sampling Interval	Buffer Size (packets)
250-10	250 ms	10
300-10	300 ms	10
250-15	250 ms	15
300-15	300 ms	15

As shown in the figures, the 300-10 experiment and 300-15 experiment exhibit similar performance for all performance metrics. Both experiments have 100% end-to-end reliability and link reliability as shown in Figure 4.14(a) and Figure 4.14(b). The average resend rate in the 300-10 experiment is 18.9%, which is only 1% different than in the 300-15 experiment. The average overhead in the 300-10 experiment is 9.5% compared with 9.3% in the 300-15 experiment. The end-to-end delay and link delay curves of the two experiments in Figure 4.12(a) and Figure 4.12(b) are almost overlapping with each other. Since no packet loss is observed in either experiment, all sensor nodes are sending at a rate matching their receiving rate. As long as buffer overflow doesn't occur during the experiment, the size of the buffer has no significant impact on the performance of the new protocol.

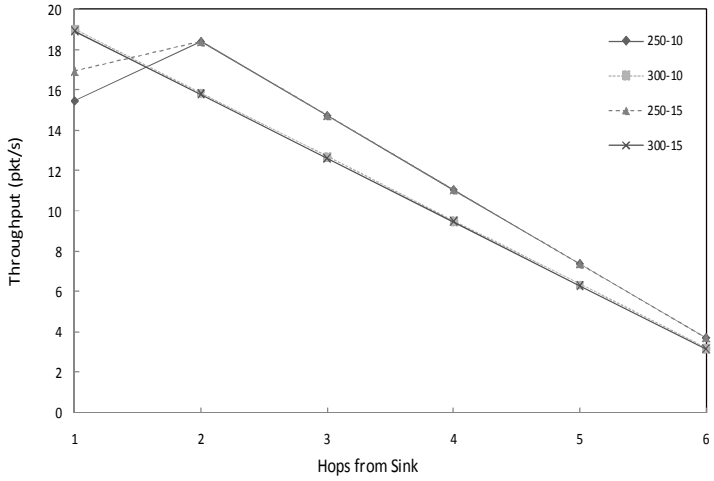
As discussed in the previous section, the network experiences packet loss when the protocol runs with 250 ms sampling interval and 50 ms sending gap because of buffer overflow. Comparing the result of the 250 ms sampling interval with 10 buffer spaces experiment (250-10) and 250 ms sampling interval with 15 buffer spaces experiment (300-15), some of the observations are as follows:



(a) End-to-end Delay

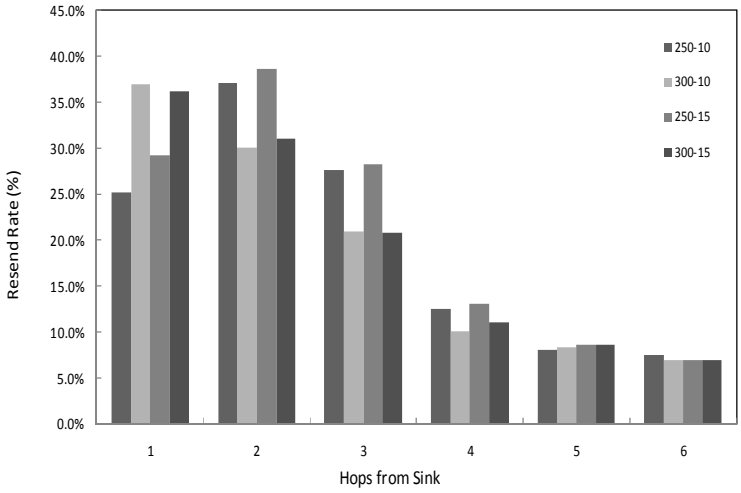


(b) Link Delay

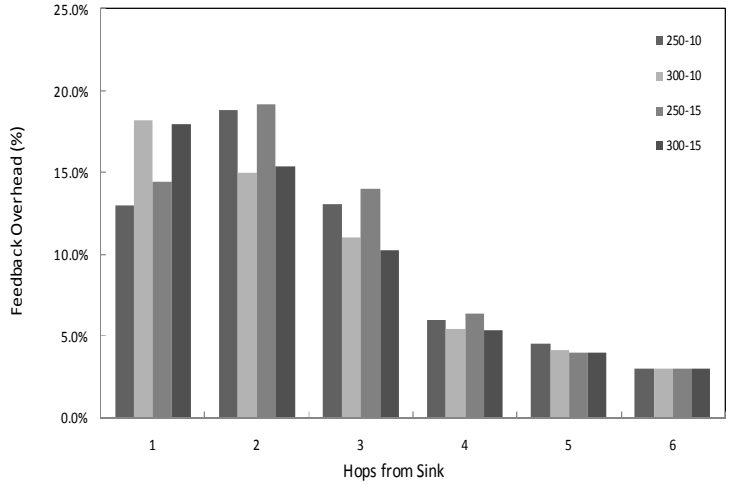


(c) Throughput

Figure 4.12 Throughput and Delay in Buffer Size Test



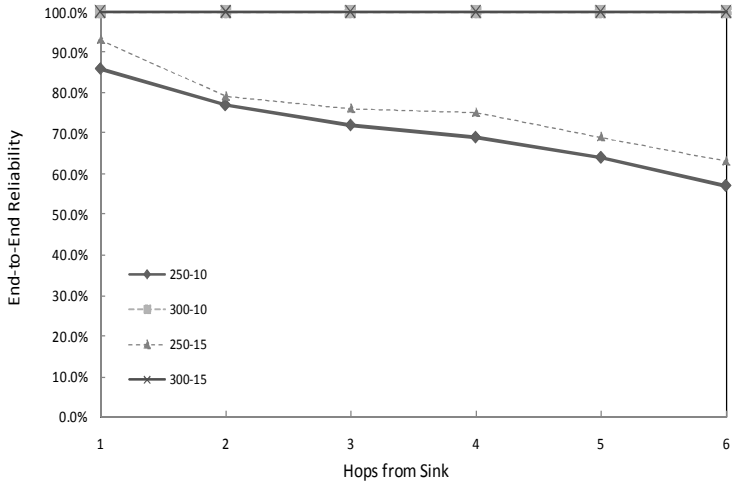
(a) Resend Rate



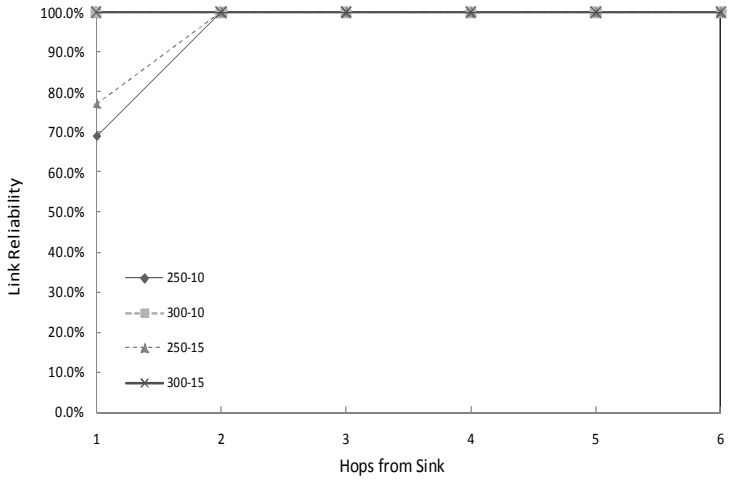
(b) Feedback Overhead

Figure 4.13 Overhead Costs in Buffer Size Test

- Due to buffer overflow, both experiments have packet loss as shown in Figure 4.14(a) and Figure 4.14(b). When a packet is missing and this packet is no longer available in the queue because of buffer overflow, the sensor node will not be able to resend the missing packet but send out the oldest packet available in its queue and sets the SKIP field in that packet. Since the 250-10 experiment has only buffer space for 10 packets compared with 15 packets in the 250-15 experiment, in the case of buffer overflow, the 250-10 experiment has a higher chance to drop packets and can recover fewer missing packets compared with the 250-15 experiment.



(a) End-to-end Reliability



(b) Link Reliability

Figure 4.14 Reliability in Buffer Size Test

Table 4.10 Results of Buffer Size Test

Experiment	End-to-End Reliability	Total Throughput (pkt/sec)	Average Resend Rate	Average Feedback Overhead
250-10	70%	15.45	19.6%	9.7%
300-10	100%	19.01	18.9%	9.5%
250-15	77%	16.92	20.8%	10.2%
300-15	100%	18.93	19.1%	9.3%

- The average end-to-end delay of the 250-15 experiment is 684 ms, which is about 25% higher than the end-to-end delay of 250-10 experiment at 552 ms. The link delay of the 250-15 experiment is also higher than the 250-15 experiment as plotted in Figure 4.12(b). The higher delay observed in the 250-15 experiment is a result of its larger buffer space. Since more data packets can be stored in the larger buffer, the chance of recovering missing packets is higher. Consequently, the average waiting time of data packets in the queue is also higher in the experiment with larger buffer size. The increasing waiting time in the transmission queue of the 250-15 experiment significantly increases the overall delay.
- The throughput of the experiment is shown in Figure 4.12(c). Since no packets are lost except of the link from node 1 to the sink, the throughput in the 250-10 experiment and in the 250-15 experiment is almost identical. However, the throughput in the 250-10 experiment at link 1 is 15.45 packets per second, which is 8.6% lower than the throughput in the 250-15 experiment at the same link. The above observation is because the higher packet loss rate at the 250-10 experiment at link 1 leads to a lower number of data packets finally received by the sink. As a result, the throughput of link 1 in the 250-10 experiment is smaller than in the 250-15 experiment.
- Finally, the results for resend rate and feedback overhead are also influenced by the buffer size. The average resend rate in the 250-10 experiment is 19.6% compared with the resend rate in the 250-15 experiment at 20.8%. The average feedback overhead of the 250-10 experiment is 9.7% compared with the overhead in the 250-15 experiment at 10.2%. In the experiment with a larger buffer space, missing packets have more retransmission opportunities to be recovered. In contrast, in the experiment with a smaller buffer space, there is a higher chance that a missing packet has already been dropped out of the queue because of buffer overflow. Thus, the retransmission opportunities for those packets are eliminated. As a result, a lower resend rate and

lower feedback overhead are observed in the experiment with smaller buffer size.

4.6 Performance Comparison with Other Protocols

In this section, the proposed new protocol is tested and compared with four other protocols with different reliability schemes. In Section 4.6.1, the new protocol is compared with a basic protocol, which doesn't implement any loss detection and recovery scheme. Section 4.6.2 presents the comparison results between the new protocol and a stop-and-wait explicit ACK protocol. Section 4.6.3 gives results for the performance differences between the new protocol and a timer-based NACK protocol. In the last section, the new protocol is compared with the modified protocol with out-of-order buffering.

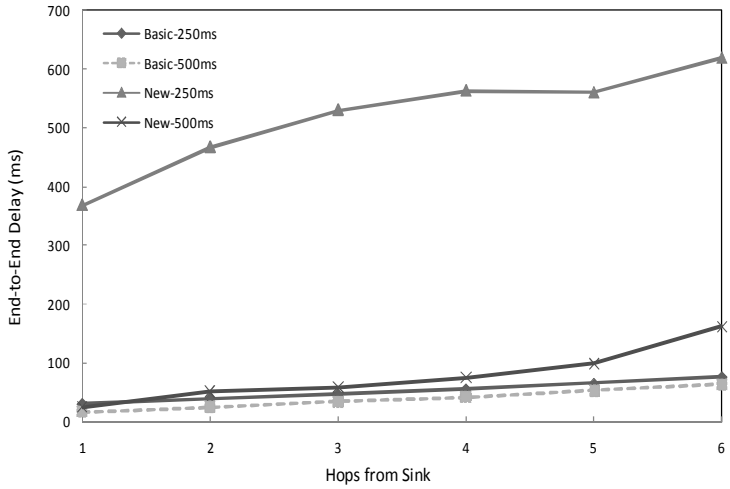
4.6.1 Comparing New Protocol with Basic Protocol

Four experiments are conducted in this section to test the performance of the new protocol and the basic protocol. The default experimental settings are used in this test, except the sampling interval varies in individual experiment. In the following, Basic-250 ms and New-250 ms refer to the basic and new protocol with a 250 ms sampling interval, respectively, while Basic-500 ms and New-500 ms refer to the basic protocol and new protocol, respectively, with a 500 ms sampling interval. The test results are shown in Figure 4.15, Figure 4.16 and Table 4.11.

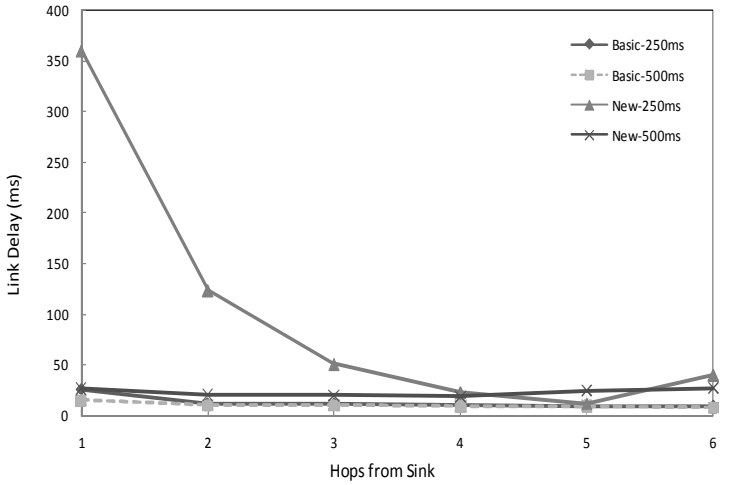
Figure 4.15(a) and Figure 4.16(b) show the end-to-end delay and the link delay results of the test, respectively. It is observed that the basic protocol with a 500 ms sampling interval achieves the lowest network delay among all four protocols. Because the basic protocol doesn't make any effort to recover the missing packets, after sending a data packet, no matter this packet is received by the receiver or not, sensor nodes in the basic protocol forward the next packet in their transmission queue. As discussed in Section 4.5.1, a 50 ms sending gap implies that the maximum sending rate at each sensor

node is 20 packets per second. With a sampling interval of 500 ms, the highest receiving rate is 12 packets per second, which occurs at node 1. Thus, when a data packet is received at a sensor node, it can be forwarded with little or no delay. The average link delay of 10ms that is observed on all but the link between node 1 and the sink can be considered as the minimum amount of time needed by a sensor node to deliver a packet in the network. In the Basic-500 ms experiment, an increasing trend of the link delay can be observed from Figure 4.15(b). This observation is due to the increasing amount of traffic as the node gets closer to the sink. In the Basic-250 ms experiment, despite the similar delay as in the Basic-500 ms experiment from node 2 to node 6, a significant increase of link delay at node 1 can be observed. In the Basic-250 ms experiment, the receiving rate is 24 packets per second at node 1, which exceeds the maximum sending rate of 20 packets per second. The packets that node 1 cannot send immediately are stored in the transmission queue and thus increase the queuing delay. When the queue is full, the new received data packets are dropped immediately. Those dropped data packets have no impact on the end-to-end delay or the link delay since they weren't sent out at all.

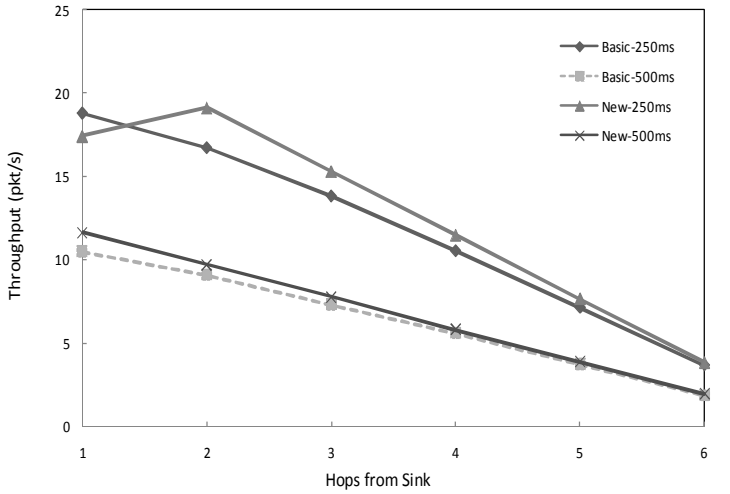
Figure 4.15(c) plots the link throughput. Both the sampling interval and the achieved reliability impact the link throughput. However, as one can observe from the results, the sampling interval plays a more important role than the reliability. The link throughput in the New-500 ms experiment is slightly higher because of higher reliability. The link throughput in the Basic-250 ms experiment and the New-250 ms experiment is about 100% higher compared with in the Basic-500 ms experiment and in the New-500 ms experiment. Given the fact that with a 250 ms sampling interval packets are generated twice as fast as with a 500 ms sampling interval, the result is reasonable. The difference between the Basic-250 ms experiment and the New-250 ms experiment is because of the difference in their link reliability. In the New-250 ms experiment, higher link reliability is achieved for all but the link between node 1 and the sink and thus it has higher throughput at those links. For the link between node 1 and the sink, however, the



(a) End-to-end Delay

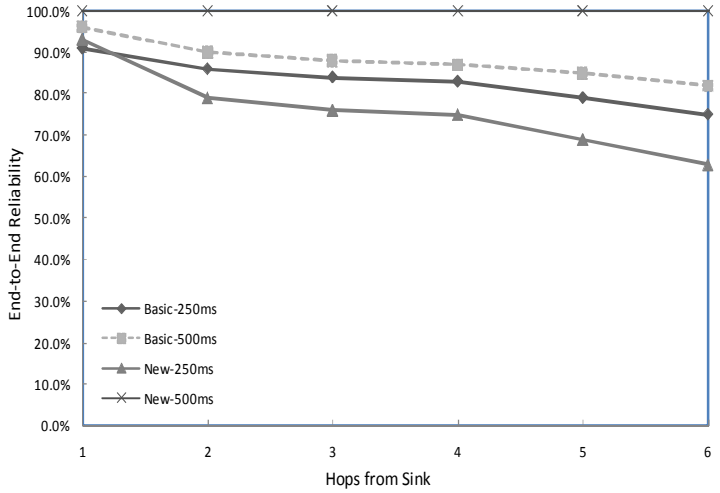


(b) Link Delay

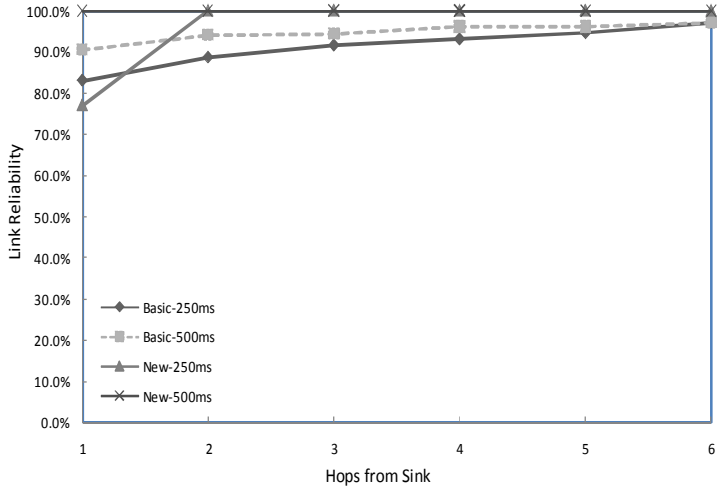


(c) Throughput

Figure 4.15 Throughput and Delay in New Protocol and Basic Protocol Test



(a) End-to-end Reliability



(b) Link Reliability

Figure 4.16 Overhead Cost in New Protocol and Basic Protocol Test

Table 4.11 Results of New Protocol and Basic Protocol Test

Experiment	End-to-End Reliability	Total Throughput (pkt/sec)	Average Resend Rate	Average Feedback Overhead
Basic-250 ms	83%	18.77	0.0%	0.0%
Basic-500 ms	91%	10.48	0.0%	0.0%
New-250 ms	77%	16.92	20.8%	10.2%
New-500 ms	100%	11.63	9.8%	4.7%

New-250 ms experiment achieves has only 77% link reliability, 8% lower than in the

Basic-250 ms experiment.

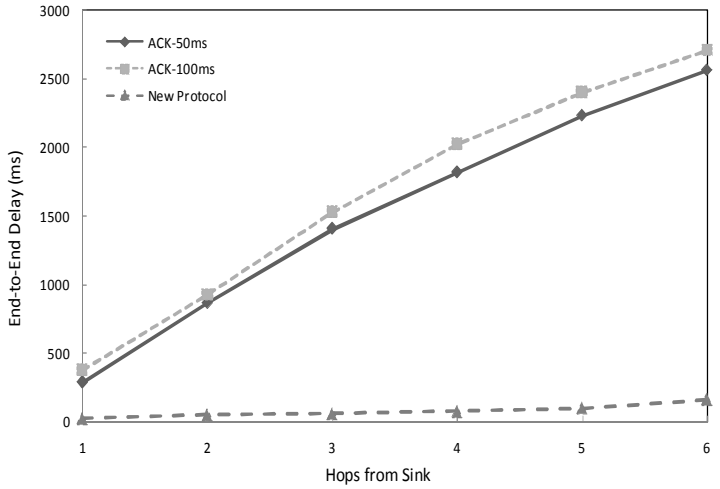
Figure 4.16(a) and Figure 4.16(b) show the end-to-end reliability and the link reliability, respectively. With 500 ms sampling interval, the new protocol, because of the hop-by-hop loss recovery scheme, achieves 100% reliability at all links. The total end-to-end reliability in the Basic-250 ms experiment is 83%, which is 9.6% lower than the total end-to-end reliability in the Basic-500 ms experiment. The Basic-250 ms experiment also exhibits lower link reliability at all links compared with the Basic-500 ms experiment. Since the basic protocol doesn't have any loss detection and loss recovery scheme, it is safe to conclude that most of the packet loss is due to corrupted packets and collisions. However, there could be another reason for the packet loss in the Basic-250 ms experiment. As analyzed in the previous paragraph, in the Basic-250 ms experiment, node 1 receives data packets at a rate higher than it can send out. When the queue is filled up, node 1 has to reject a number of packets and discard them right away. The new protocol surprisingly underperforms the basic protocol on all performance metrics. The overall end-to-end reliability is 77% in the New-250 ms experiment compared with 83% in the Basic-250 ms experiment. The average end-to-end delay in the New-250 ms is 517ms compared with 51ms in the Basic-250 ms experiment. The total throughput in the New-250 ms experiment is 17.4 packets per second, while the total throughput is 18.8 packets per second in the Basic-250 ms experiment. The link delay at node 1 is around 360ms in the New-250 ms experiment compared with 26ms in the Basic-250 ms experiment. In the New-250 ms experiment, a large number of feedback packets are injected into the network (around 10.2%) relative to the total number of data packets, which triggers many packet resends (around 20.8%). The above comparison results show that, even through the hop-by-hop recovery scheme in the new protocol helps to repair some of the packet loss caused by packet corruptions, the feedback packets and resent packets generated by the recovery scheme may actually lead to a higher possibility of packet collisions and channel contentions, which in turn results in significant delay and larger loss rate.

4.6.2 Comparing New Protocol with ACK Protocol

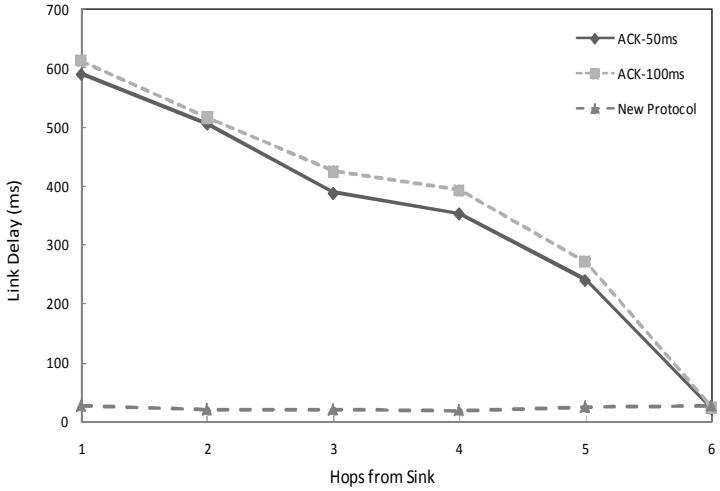
In this section, the new protocol is tested and compared with the ACK-based reliable protocols. In this section, three experiments are conducted. The first experiment tests the performance of an ACK protocol with a 50 ms ACK timer. The second experiment tests the performance of the same ACK protocol with a 100 ms ACK timer. The last experiment tests the performance of the new protocol. The default experimental settings with 500 ms sampling interval are used in this test. The test results are shown in Figure 4.17, Figure 4.18 and Table 4.12.

Figure 4.17(a) plots the end-to-end delay of the experiments. In the ACK protocol, because the new data packet cannot be sent until the previous sent packet was acknowledged by the receiver, tremendous queuing delay is introduced to the end-to-end delay of the packet. Both ACK protocols show significant delay compared with the new protocol. The average end-to-end delay in the ACK-50 ms experiment is 1529.55ms and the average end-to-end delay in the ACK-100ms experiment is 1660.40 ms. The new protocol, on the other hand, allows the transmission of the new data packet in parallel with the loss detection and recovery process. Thus, the average end-to-end delay of the New Protocol experiment is only 78.29 ms. A similar result can be observed in the link delay as shown in Figure 4.17(b). The new protocol shows much smaller delay than the ACK protocols at all links. The advantage of the new protocol over the ACK protocol in terms of delay is obvious. It is worth mentioning that the delay in the ACK-50 ms experiment is lower than in the ACK-100 ms experiment as observed because the resend timer of the ACK-50 ms experiment is smaller. As a result, in the case where the resend timer is fired and the sender needs to retransmit the missing packet, the queuing delay in the ACK-50 ms experiment is relatively smaller.

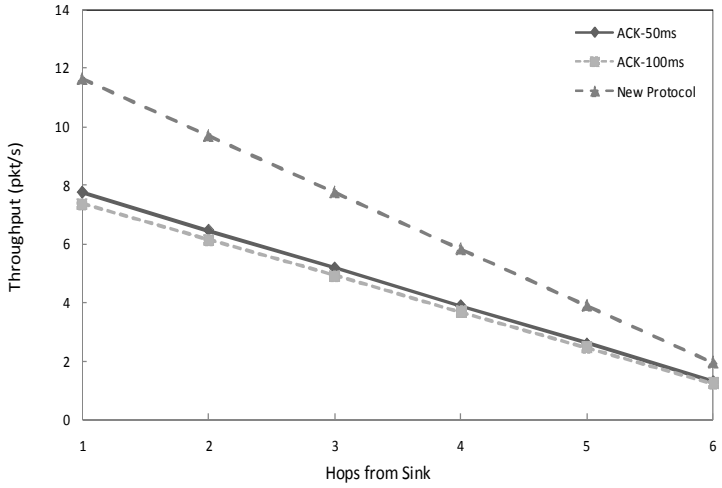
The resend rate plotted in Figure 4.18(a) exhibits some variance among the three experiments. As explained in the previous section, the new protocol employs a NACK-



(a) End-to-end Delay

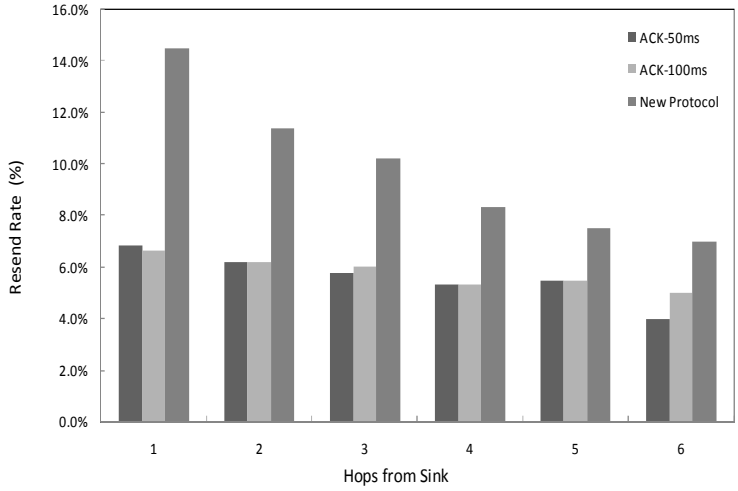


(b) Link Delay

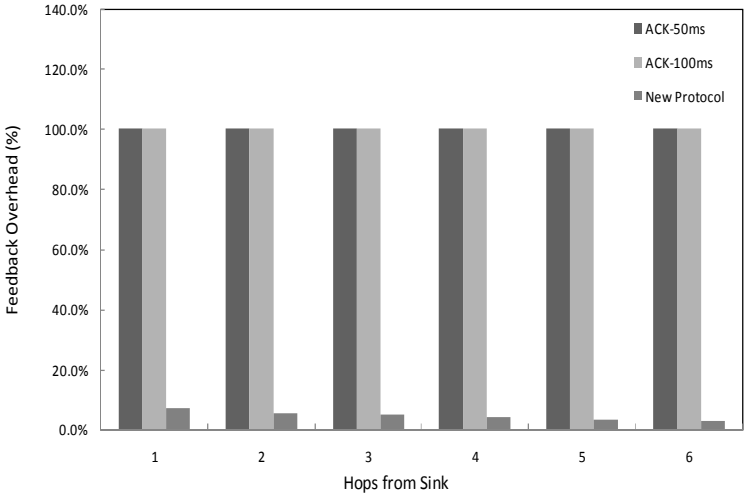


(c) Throughput

Figure 4.17 Throughput and Delay in New Protocol and ACK Protocol Test



(a) Resend Rate



(b) Feedback Overhead

Figure 4.18 Overhead Cost in New Protocol and ACK Protocol Test

Table 4.12 Results of New Protocol and ACK Protocol Test

Experiment	End-to-End Reliability	Total Throughput (pkt/sec)	Average Resend Rate	Average Feedback Overhead
ACK-50 ms	100%	7.74	5.7%	100.0%
ACK-100ms	100%	7.35	5.8%	100.0%
New Protocol	100%	11.63	9.8%	4.7%

based loss detection approach where packet loss can only be detected when the receiver receives another data packet with gap in the packet ID. As a result, for every missing

packet, both the missing packet and the packet next to that packet in the transmission queue have to be retransmitted. However, in the ACK-based protocol, since the loss detection is associated with a local resend timer, the sender only needs to resend the missing packet. As one can observe from Figure 4.18(a), the average resend rate in the New Protocol experiment is 9.8%, whereas the resend rate in the ACK-50 ms experiment is 5.7% and resend rate in the ACK-100ms experiment is 5.8%.

Figure 4.18(b) presents the result of overhead of the test. Since the receiver in the ACK protocol is responsible to create ACK packet and confirm the reception of every data packet, the overhead of the ACK protocol is 100%. In the new protocol, however, the receiver only needs to reply a NACK when a missing packet is identified. The average overhead of NACK is 4.7%, which is much smaller compared with ACK protocols. Small overhead is the main reason that a NACK-based approach is usually preferred over an ACK-based approach in wireless sensor networks.

All experiments in this test show 100% reliability. Although all data packets are received by the sink at all experiments, the duration of the each experiment varies. Because of the variance of the resend timer and the queuing delay, the ACK-50 ms experiment lasts 77368 ms, the ACK-100 ms experiment lasts 81630ms, while the New Protocol experiment last only 51569 ms. As a result, the New Protocol experiment shows the total throughput of 11.63 packets per second, 49% higher than in the ACK-50 ms experiment and 57% higher than in the ACK-100ms experiment.

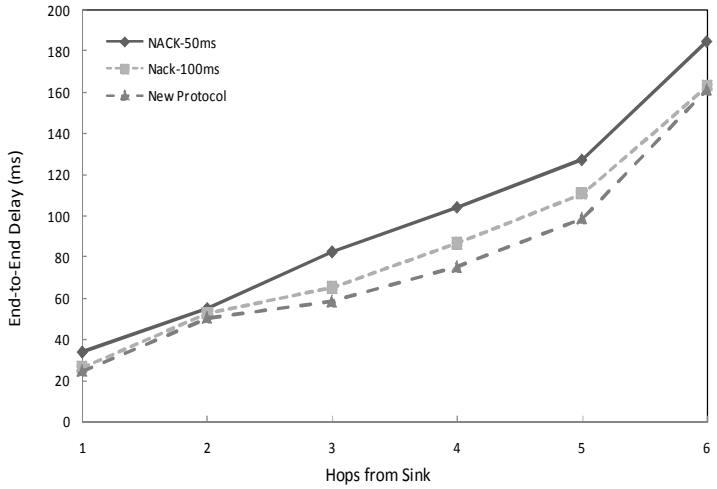
4.6.3 Comparing New Protocol with NACK Protocol

In this section, three experiments are conducted to compare the performance of the new protocol and a timer-based NACK protocol. The NACK-50 ms represents the NACK protocol with a 50 ms NACK timer, and the NACK-100 ms represents the NACK protocol with a 100 ms NACK timer. The default experimental settings with a 500 ms sampling interval are used in this test. The test results are shown in Figure 4.19, Figure

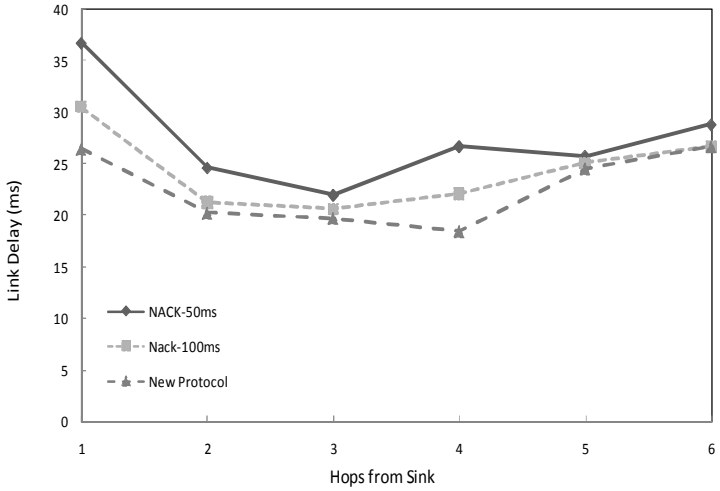
4.20 and Table 4.13.

50 ms and 100 ms are chosen as the length of the NACK timer in this test because the sending gap at all sensor nodes is set to 50 ms. If the timer is reduced to a value smaller than 50 ms, it is possible that a retransmission has already been scheduled and is waiting to be sent. The receiver, however, may assume that the NACK or the retransmission was lost when the timer expires and thus will send out a redundant NACK packet. The timer-based NACK protocol is expected to reduce the network delay at the expense of a higher overhead rate.

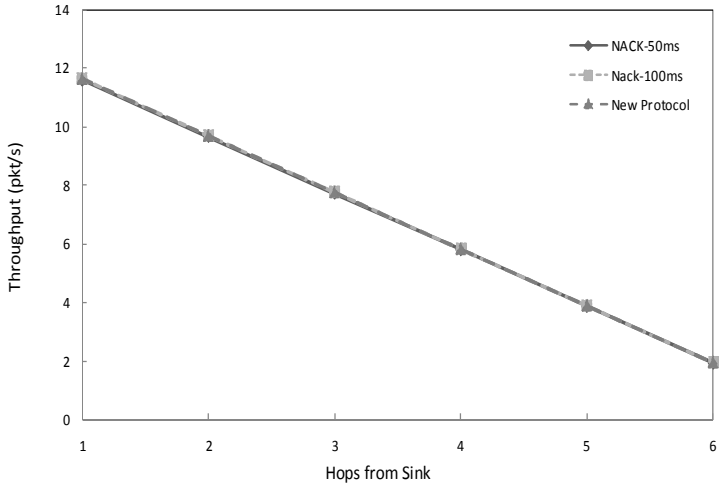
As shown in Figure 4.19 and Table 4.13, all three protocols achieve 100% end-to-end reliability and have similar throughput. In Figure 4.20(b), the timer-based NACK protocols show higher overhead. The average overhead in the NACK-50 ms experiment is 12.5%, which is 20.2% higher than in the NACK-100 ms experiment and 27.5% higher than in the New Protocol experiment. However, as plotted in Figure 4.19(a) and Figure 4.19(b), the timer-based NACK protocols have even higher delay than the new protocol which has no NACK timer. The average end-to-end delay in the NACK-50 ms experiment, NACK-100ms experiment and New Protocol experiment is 97.94 ms, 84.18 ms and 78.29 ms, respectively. The higher delay of the timer-based NACK protocol is likely because, although aggressively sending NACKs and requesting retransmissions may reduce queuing delay for some data packets, the additional overhead generated may cause a higher possibility of packet corruptions and channel contention. In fact, the consecutive loss of packets in the scenarios described above is possibly caused by local network congestion. In the timer-based NACK approach, by sending additional NACK packets, the nodes are injecting more packets into a congested network, which may further aggravate the network congestion. In Figure 4.20(a), the NACK-50 ms experiment shows a higher resend rate than the other two experiments. One may draw the conclusion that the implementation of a NACK timer leads to higher numbers of NACK packets, which results in higher numbers of retransmissions and finally impacts delay.



(a) End-to-end Delay

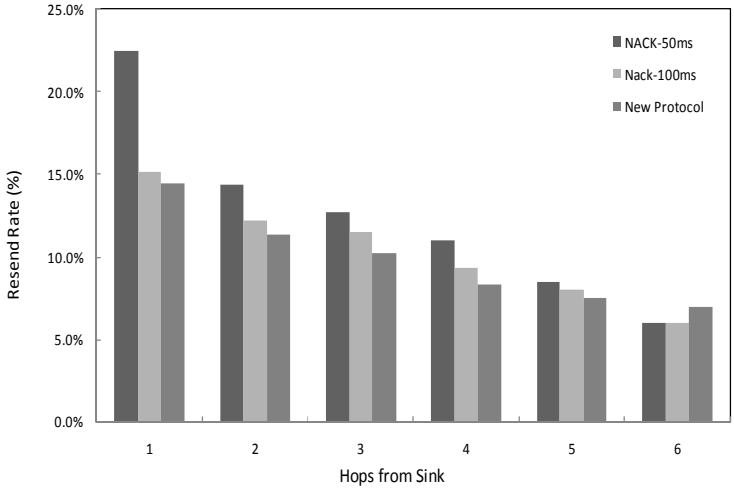


(b) Link Delay

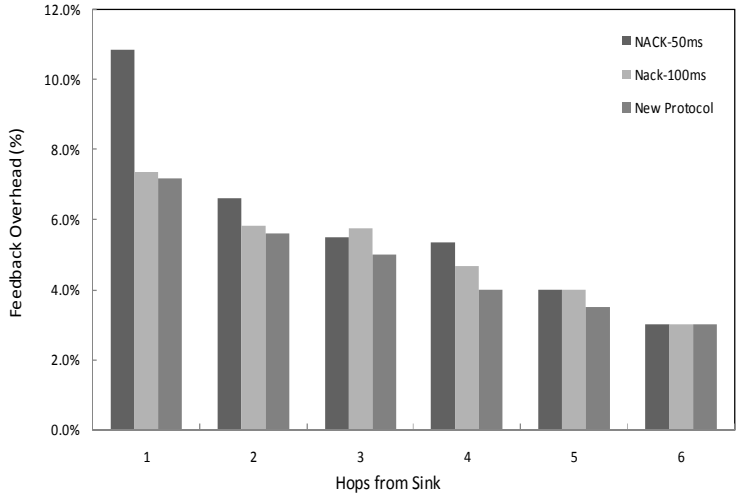


(c) Throughput

Figure 4.19 Throughput and Delay in New Protocol and NACK Protocol Test



(a) Resend Rate



(b) Feedback Overhead

Figure 4.20 Overhead Cost in New Protocol and NACK Protocol Test

Table 4.13 Results of New Protocol and NACK Protocol Test

Experiment	End-to-End Reliability	Total Throughput (pkt/sec)	Average Resend Rate	Average Feedback Overhead
NACK-50 ms	100%	11.62	12.5%	5.9%
NACK-100 ms	100%	11.62	10.4%	5.1%
New Protocol	100%	11.63	9.8%	4.7%

Compared with the NACK-50 ms experiment, the performance results in the NACK-100 ms experiment are closer to the result of New Protocol experiment. For example, the

average overhead and average resend rate in the NACK-100ms experiment is only 6% larger than in the New Protocol experiment and it is about 20% smaller than in the NACK-50 ms experiment. A similar trend can be observed in the results of the end-to-end delay and the link delay. The reason for the above observation is that, when increasing the NACK timer to 100 ms, which is twice as large as the sending gap, the chance of receiving no replies at the receiver decreases significantly. Most of the NACK timers were suppressed and only a very small number of retransmissions were triggered. Thus, the NACK-100 ms experiment shows similar results as in the New Protocol experiment.

4.6.4 Comparing New Protocol with Modified Protocol

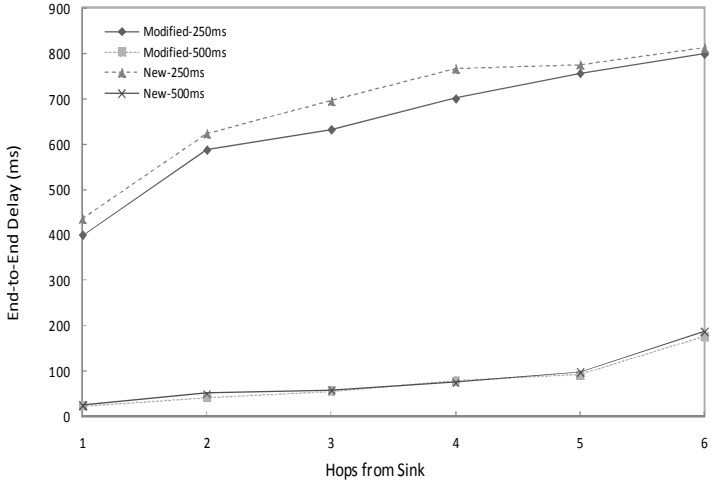
As described in Section 3.6.4, a modified protocol with out-of-order buffering is proposed in this work for the purpose of performance enhancement. Four experiments are conducted in this test to test the performance of the original protocol and the modified protocol. The default experimental settings are used in this test, except the sampling rate varies in individual experiments. New-250 ms and Modified-250 ms represent the new protocol and the modified protocol with 250 ms sampling interval, respectively, and New-500 ms and Modified-500 ms represent the new protocol and the modified protocol with 500 ms sampling interval, respectively. The experimental results are shown in Figure 4.21, Figure 4.22, Figure 4.23 and Table 4.14.

Figure 4.21(a) and Figure 4.21(b) plot the end-to-end delay and the link delay observed in the experiments. When using a 500 ms sampling interval, both protocols show very similar network delay. Although the modified protocol is able to recover out-of-order packets from its local buffer rather than requesting additional retransmissions, the amount of time those packets spend in the out-of-order buffer becomes part of their network delay. However, when the network starts to experience packet loss, when using a 250 ms sampling interval, the modified protocol shows better delay results at both end-to-end level and link level than the original protocol. The lower delay of the modified

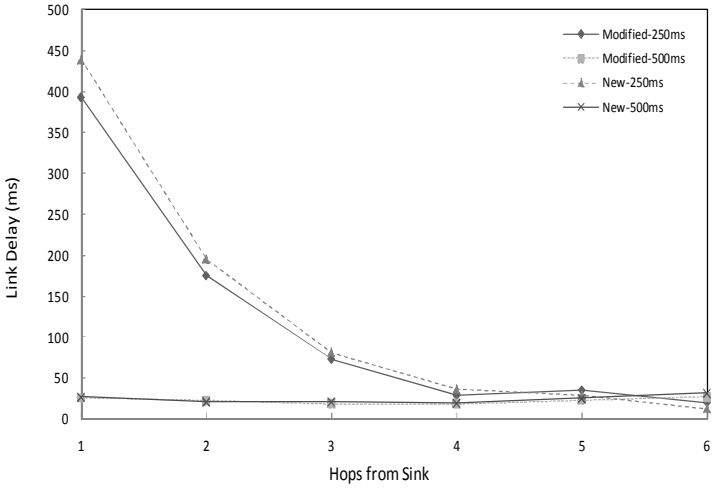
protocol is a consequence of the implementation of the out-of-order buffer. It is reasonable to assume that when the wireless channel becomes lossy and congested, the NACK packet as well as the retransmissions may also be lost during their transmission. Since the modified protocol stores an out-of-order packet in its own buffer, it is able to recover that packet immediately after receiving the correct in-order packet. For the original protocol, however, the receiver relies on retransmission from the sender to recover the missing packet. If retransmission isn't successful because of the increasing level of channel contention and packet collisions, multiple retransmissions may be needed for one missing packet. As a result, the accumulated queuing delay eventually increases the network delay of the original protocol.

The throughput of both of the 500 ms sampling interval experiments are almost identical as plotted in Figure 4.21(c). The Modified protocol with 250 ms sampling interval, however, exhibits higher throughput at link 1 compared with the original protocol. The variance of the throughput is a result of the difference in the end-to-end reliability. Because of the implementation of the out-of-order buffer, which results in a lower resend rate, the modified protocol shows 79% end-to-end reliability, which is 2% higher than the original protocol.

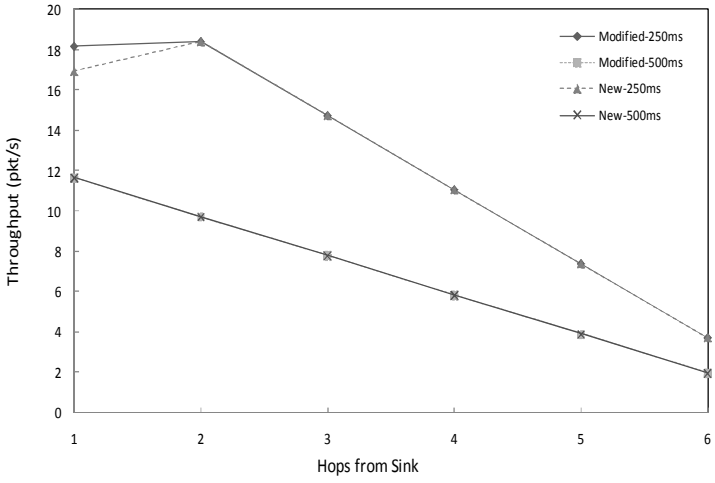
From Figure 4.22(a), one can observe that the resend rate in the Modified-500 ms experiment is much lower than in the New-500 ms experiment. The average resend rate in the Modified-500 ms experiment is only 4.9% compared with 9.9% in the New-500 ms experiment. As explained in the previous section, the above observation is because the receiver in the modified protocol is able to recover some of the out-of-order packets from its local buffer and thus the sender can skip the retransmissions of those packets. When using a 250 ms sampling interval, the variance of the resend rate between protocols still exists but narrows. The average resend rate in the Modified-250 ms experiment is 12.8% compared with 20.8% in the New-250 ms experiment. Part of the reason for the above result is that the network is likely to experience more consecutive



(a) End-to-end Delay

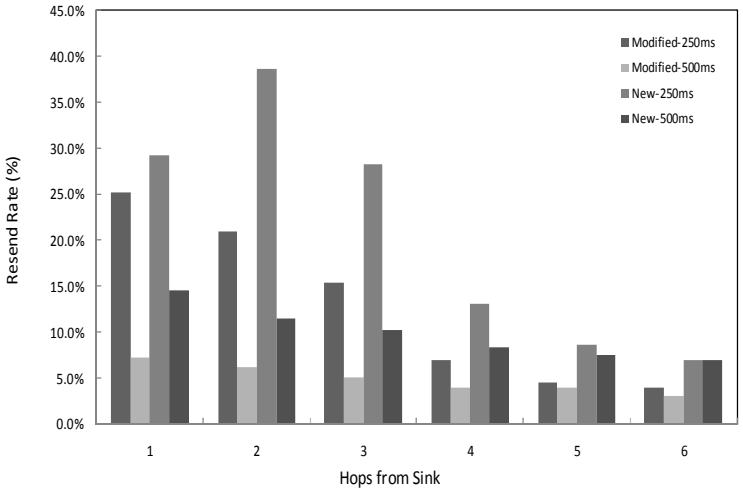


(b) Link Delay

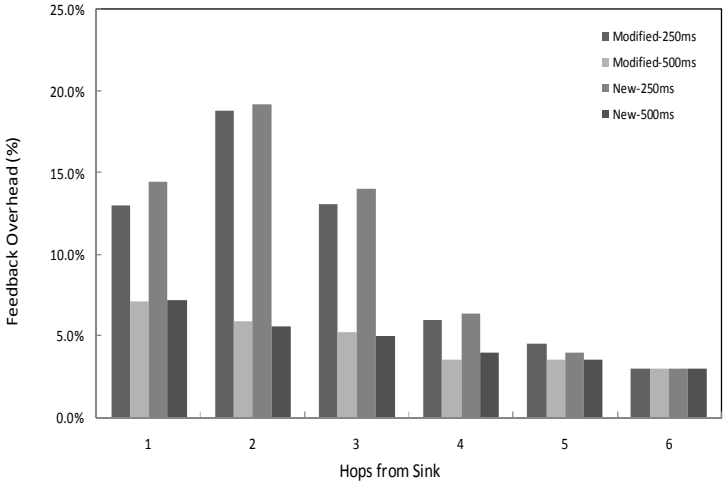


(c) Throughput

Figure 4.21 Throughput and Delay in New Protocol and Modified Protocol Test



(a) Resend Rate

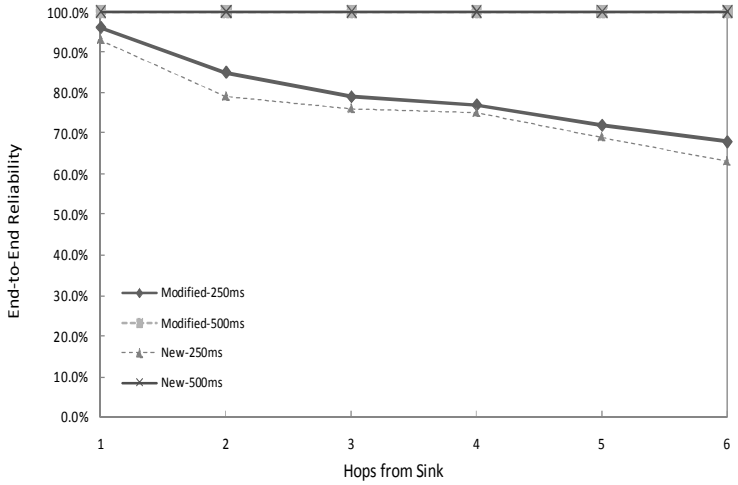


(b) Feedback Overhead

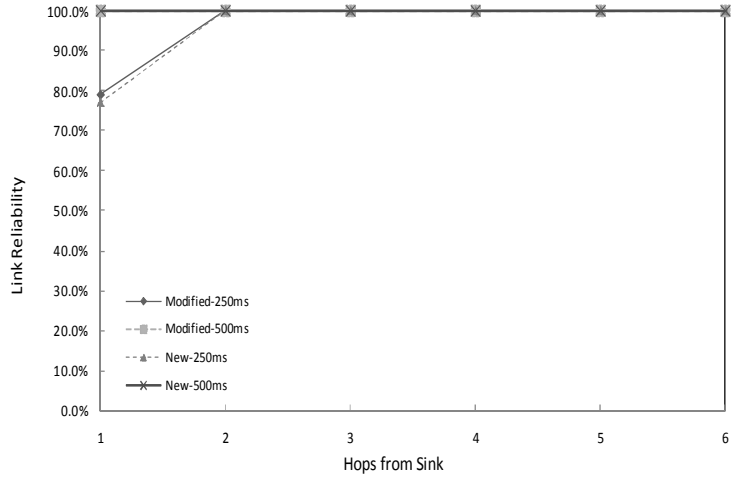
Figure 4.22 Overhead Costs in New Protocol and Modified Protocol Test

packet loss with the 250 ms sampling interval. In this case, even in the modified protocol, the sender may still need to resend all of the missing packets.

The overhead of the experiments is plotted in Figure 4.22(b). Protocols with 500 ms sampling interval show higher overhead compared to protocols with 250 ms sampling interval. Comparing the two protocols with the same sampling interval, the variance is minor. The average overhead in the Modified-250 ms experiment is 9.7% compared with 10.2% in the New-250 ms experiment; the average overhead in the Modified-500 ms experiment is 4.7% compared with 4.8% in the New-500 ms experiment.



(a) End-to-end Reliability



(b) Link Reliability

Figure 4.23 Reliability in New Protocol and Modified Protocol Test

Table 4.14 Results of New Protocol and Modified Protocol Test

Experiment	End-to-End Reliability	Total Throughput (pkt/sec)	Average Resend Rate	Average Feedback Overhead
Modified-250 ms	79%	18.16	12.8%	9.7%
Modified-500 ms	100%	18.97	4.9%	4.7%
New-250 ms	77%	16.92	20.8%	10.2%
New-500 ms	100%	18.93	9.9%	4.8%

The results for the end-to-end reliability and the link reliability are shown in Figure

4.23 (a) and Figure 4.23(b). Only the experiments with 250 ms sampling interval experienced packet loss. The reason for the packet loss, as explained in Section 4.5.1, is because the sending rate is smaller than the receiving rate at node 1. When the transmission queue is filled up, sensor nodes are forced to drop packets. The out-of-order buffer implemented in the modified protocol helps to improve the reliability. The end-to-end reliability in the modified-250 ms experiment is 79%, compared with 77% in the New-250 ms experiment. The modified protocol is able to recover some of the missing packets from its local buffer and thus avoid the retransmission of those packets.

CHAPTER 5

CONCLUSIONS

This thesis studied the reliable data delivery issue in wireless sensor networks. A NACK-based hop-by-hop reliable transport layer protocol is developed and evaluated in this work. This chapter summarizes the work that has been done in this thesis and discusses some directions for future work. Section 5.1 provides a brief summary of the thesis. Section 5.2 states the contributions of this work. In Section 5.3, some of the possible future work is described.

5.1 Thesis Summary

This thesis addresses the reliability issues in data transport in wireless sensor networks. The design goal was to provide a solution that is able to maintain 100% reliable data delivery (except in the case of buffer overflow) with minimal delay and overhead. Chapter 2 provides an overview of the current research on reliable data transport in wireless sensor networks as well as describing some existing data transport protocols. A new hop-by-hop reliable data delivery protocol is proposed in Chapter 3. The new protocol is designed based on a NACK loss detection and recovery scheme. A timer-based explicit ACK approach is also used to address the last/single packet delivery problem. The new queue management scheme designed in the protocol is used to efficiently schedule the data transmission and retransmission. The performance of the new protocol is tested in a Crossbow MicaZ testbed. Performance results are given in Chapter 4. Ten separate tests of the protocol are conducted to evaluate the performance of

the protocol. The first group of tests including the traffic test, the scalability test, the sampling interval test and the interference test, demonstrates some basic properties of the protocol under various system and protocol parameters. The second group of tests including the test of sending gap with sampling interval and the test of buffer size, illustrates the performance of the protocol with some extreme parameter settings. The last group of tests compares the performance of the new protocol to the performance of four other protocols including a basic protocol with no packet loss recovery mechanisms, an explicit stop-and-wait ACK protocol, a timer-based NACK protocol and the modified new protocol with out-of-order buffering.

5.2 Contributions

The main contribution of this thesis is the design and the evaluation of a hop-by-hop reliable data delivery protocol. In particular, the contributions are as follows:

- The general issues in designing a reliable data transport protocol for wireless sensor networks are discussed. A survey is conducted of some of the existing data transport protocols focusing on reliability and congestion control.
- A NACK-based loss detection and recovery scheme is designed for reliable data delivery in wireless sensor networks.
- A solution is provided to the last/single packet delivery problem in the conventional NACK-based approach by introducing a timer-based explicit ACK approach to the new protocol.
- A new queue management scheme is designed. This scheme gives priority to fresh data, which is preferable in some WSN applications. Nodes with the new queue management scheme are able to transmit new data packets in parallel with the detection and recovery of missing packets.
- A variant of the new protocol that buffers out-of-order packets is designed. Experimental results show that the modified protocol performs better in some

conditions compared with the original new protocol.

- The new protocol is implemented and tested in a MicaZ testbed under various system and protocol parameter settings. The new protocol is also tested and compared with four other protocols.

5.3 Future Work

The new protocol proposed in this work is evaluated in a MicaZ testbed and proven to be able to provide 100% reliability (except under some extreme conditions) and reduce overhead and delay. However, there are some limitations of this work that can be improved in the future. First, the evaluation considers only single line topology with one destination (the sink). Some further tests can be done with a more general topology setting such as with multiple source nodes and multiple destination nodes. Second, as discussed in Section 3.6.2, route changes because of node failure or network congestion are not uncommon in wireless sensor network applications. Route changes may also result in the transmission of redundant data packets by the sender. A test of the new protocol with route changes could be useful to further evaluate the performance of the new protocol. Third, the tests conducted in Chapter 4 use only a small number of sensor nodes. A larger scale test of the protocol with a greater number of sensor nodes may be desirable as part of the future work. Fourth, fairness among different traffic flows could be considered in the future design. Last, the new protocol could be incorporated with other MAC layer, routing layer or network layer protocols. The cross-layer design of a reliable data transport protocol is attractive.

REFERENCES

[1] I. F. Akyildiz, W. Su, Y. Sankarasubramaniam, and E. Cayirci. A survey on sensor networks. *Communications Magazine, IEEE*, 40(8):102--114, 2002.

[2] CrossBow Technology. http://www.xbow.com/

[3] B. Deb, S. Bhatnagar, and B. Nath. ReInForM: Reliable Information Forwarding Using Multiple Paths in Sensor Network. In *Proc. LCN '03*, Konigswinter, Germany, Oct. 2003, pp. 406--415.

[4] A. Dunkels, J. Alonso, T. Voigt and H. Ritter. Distributed TCP Caching for Wireless Sensor Networks. In *Proc. 3rd Annual Mediterranean Ad Hoc Net. Workshop*, Bodrum, Turkey, June 2004, pp. 21--31.

[5] C.-T. Ee and R. Bajcsy. Congestion Control and Fairness for Many-to-One Routing in Sensor Networks. In *Proc. ACM Sensys '04*, Baltimore, MD, Nov. 2004, pp.148--161.

[6] D. Estrin. Reliability and Storage in Sensor Networks. Technical Report in CENS, UCLA, Los Angeles, CA, 2005.

[7] E. Felemban, C. Lee, E. Ekici, R. Boder, and S. Vural. Probabilistic QoS guarantee in reliability and timeliness domains in wireless sensor networks. In *Proc. IEEE INFOCOM '05*, Miami, FL, Mar. 2005, pp. 2646--2657.

[8] S. Gobriel, S. Khattab, D. Mosse, J. Brustoloni, and R. Melhem. RideSharing: Fault Tolerant Aggregation in Sensor Networks Using Corrective Actions. In *Proc. SECON '06,* Reston, VA, Sept. 2006, pp. 595--604.

[9] Y. Gu and T. He. Data Forwarding in Extremely Low Duty-Cycle Sensor Networks with Unreliable Communication Links. In *Proc. ACM SenSys '07*, Sydney, Australia, Nov. 2007, pp. 321--334.

[10] C. Gui and P. Mohapatra. Power conservation and quality of surveillance in target tracking sensor networks. In *Proc. IEEE MobiCom '04*, Philadelphia, PA, Sept. 2004, pp. 129--143.

[11] T. He, John A. Stankovic, C. Lu, and T. Abdelzaher. SPEED: A stateless protocol for real-time communication in sensor networks. In *Proc. ICDCS '03*, Providence, RI, May 2003, pp. 46--55.

[12] J. Heidemann and R. Govindan. An overview of embedded sensor networks. Technical report, USC/Information Sciences Institute, Nov. 2004.

[13] J. Hui and D. Culler. The dynamic behavior of a data dissemination algorithm at scale. In *Proc. ACM Sensys '04*, Baltimore, MD, Nov. 2004, pp.59--71.

[14] B. Hull, K. Jamieson, and H. Balakrishnan. Mitigating Congestion in Wireless Sensor Networks. In *Proc. ACM SenSys '04,* Baltimore, MD, Nov. 2004, pp. 134--147.

[15] C. Intanagonwiwat, R. Govindan, D. Estrin, J. Heidemann, and F. Silva. Directed Diffusion for Wireless Sensor Networking. *ACM/IEEE Transactions on Networking*, 11(1):2--18, Feb. 2002.

[16] Y. G. Iyer, S. Gandham, and S. Venkatesan. STCP: A Generic Transport Layer Protocol for Wireless Sensor Networks. In *Proc. IEEE ICCCN '05*, San Diego, CA, Oct. 2005, pp. 449--454.

[17] H. Karl and A. Willig. *Protocols and Architectures for Wireless Sensor Networks.* John Wiley & Sons, 2005.

[18] S. Kim, R. Fonseca, P. Dutta, A. Tavakoli, D. Culler, P. Levis, S. Shenker, and I. Stoica. Flush: A Reliable Bulk Transport Protocol for Multihop Wireless Networks. In *Proc. ACM SenSys '07*, Sydney, Australia, Nov. 2007, pp. 351--365.

[19] H. Lee, Y. Ko, and D. Lee. A Hop-by-hop Reliability Support Scheme for Wireless Sensor Networks. In *Proc. PERCOMW '06,* Pisa, Italy, Mar. 2006, pp. 431--439.

[20] C. Miller and C.Poellabauer. PALER: A Reliable Transport Protocol for Code Distribution in Large Sensor Networks. In *Proc. SECON '08,* San Francisco, CA, Jun. 2008, pp. 206--214.

[21] J. Paek and R. Govindan. RCRT: Rate-Controlled Reliable Transport for Wireless Sensor Networks. In *Proc. ACM SenSys '07*, Sydney, Australia, Nov. 2007, pp. 305--319.

[22] S. Park, R. Vedantham, R. Sivakumar, and I. Akyildiz. A Scalable Approach for Reliable Downstream Data Delivery in Wireless Sensor Networks. In *Proc. MobiHoc '04*, Tokyo, Japan, May 2004, pp. 78--89.

[23] V. Raghunathan, C. Schurgers, S. Park, M. Srivastava, and B. Shaw. Energy-aware wireless microsensor networks. *IEEE Signal Processing Magazine*, 19(2):40--50, 2002.

[24] N. Rahnavard, and F. Fekri. CRBcast: A Collaborative Rateless Scheme for Reliable and Energy-Efficient Broadcasting in Wireless Sensor Networks. In *Proc. ISPN '06,* Nashville, TN, April 2006, pp. 276--283.

[25] V. Rajendran, K. Obraczka, and J. J. Garcia-Luna-Aceves. Energy-efficient collision-free medium access control for wireless sensor networks. In *Proc. ACM SenSys '03,* Los Angeles, CA, Nov. 2003, pp. 181--192.

[26] F. Rocha. A. Grilo, P. R. Pereria, M. S. Nunes and A. Casaca. Performance Evaluation of DTSN in Wireless Sensor Networks. In *Proc. EuroNGI '08,* Barcelona, Spain, Jan. 2008, pp. 1--9.

[27] Z. Rosberg, R.P. Liu, A. Y. Dong, L. D. Tuan, and S. Jha. ARQ with Implicit and Explicit ACKs in Sensor Networks. In *Proc. IEEE Globecom '08,* New Orleans, LA, Dec. 2008, pp. 1--6.

[28] Y. Sankarasubramaniam, O. Akan, and I. Akyildiz. ESRT: Event-to-Sink Reliable Transport in Wireless Sensor Networks. In *Proc. MobiHoc '03,* Annapolis, MD, June 2003, pp. 177--188.

[29] F. Stann, and J. Heidemann. RMST: Reliable Data Transport in Sensor Networks. In *Proc. SNPA '03*, Anchorage, AK, June 2003, pp. 102--112.

[30] The TinyOS. http://www.tinyos.net/

[31] R. Vedantham, R. Sivakumar, and S. Park. Sink-to-Sensors Congestion Control. In *Proc. ICC '05,* May 2005, pp. 3211--3217.

[32] C.-Y. Wan, A. Campbell, and L. Krishnamurthy. Pump-Slowly, Fetch-Quickly (PSFQ): A Reliable Transport Protocol for Sensor Networks. In *Proc. WSNA '02*, Atlanta, GA, Sept. 2002, pp. 1--11.

[33] C.-Y. Wan, S. Eisenman, A. Campbell and J. Crowcroft. Siphon: Overload Traffic Management Using Multi-Radio Virtual Sinks in Sensor Networks. In *Proc. ACM SenSys '05*, San Diego, CA, Nov. 2005, pp. 116--129.

[34] C.-Y. Wan. S. B. Eisenman, and A. T. Campbell. CODA: Congestion Detection and Avoidance in Sensor Networks. In *Proc. ACM SenSys '03,* Los Angeles, CA, Nov. 2003, pp. 266--279.

[35] L. Wang, and S. Kulkarni. Proactive Reliable Bulk Data Dissemination in Sensor Networks. In *Proc. PDCS '05,* Phoenix, AZ, Nov. 2005, pp. 773--778.

[36] C. Wang, K. Sohraby, V. Lawrence, B. Li and Y. Hu. Priority-Based Congestion Control in Wireless Sensor Networks. In *Proc. STCU '06,* Taichung, Taiwan, June, 2006, pp. 21--31.

[37] Y. Wang, M. Martonosi, and L.-S. Peh. Supervised Learning in Sensor Networks: New Approaches with Routing, Reliability Optimizations. In *Proc. SECON '06,* Reston, VA, Sept. 2006, pp. 256--265.

[38] A. Woo and D. C. Culler. A Transmission Control Scheme for Media Access in Sensor Networks. In *Proc. ACM Mobicom '01*, Rome, Italy, July 2004, pp. 221--235.

[39] W. Ye, J. Heidemann, and D. Estrin. Medium access control with coordinated adaptive sleeping for wireless sensor networks. *IEEE/ACM Transactions on Networking*, 12(3):493--506, Dec. 2004.

[40] J. Yicka, B. Mukherjeea, and D. Ghosal. Wireless sensor network survey. *Computer Networks*, 52(12):2292–2330, 2008.

[41] B. Yu, P. Scerri, K. Sycara, Yang Xu, and M. Lewis. Scalable and Reliable Data Delivery in Mobile Ad Hoc Sensor Networks. In *Proc. AAMAS '06*, Hakodate, Japan, May 2006, pp. 71--83.

[42] H. Zhang, A. Arora, Y.-R. Choi, and M. Gouda. Reliable Bursty Convergecast in Wireless Sensor Networks. In *Proc. MobiHoc '05*, Urbana-Champaign, IL, May 2005, pp. 266--276.

[43] Y. Zhou, M. R. Lyu, J. Liu, and H. Wang. PORT: A Price-Oriented Reliable Transport Protocol for Wireless Sensor Networks. In *Proc. ISSRE '05,* St. Malo, France, Nov. 2005, pp. 10--23.